DeSalvo Standard Library

Generated by Doxygen 1.8.3.1

Fri Dec 16 2016 16:47:10

Contents

Chapter 1

DeSalvo Standard Library Documentation

A core library for C++ applications

Author

Stephen DeSalvo

Date

July 2015 The namespace name is desalvo_standard_library, in honor to the Standard Library (NOT STL) but expanded to include my versions of commonly used functionality. It will be abbrev. to dsl in the future, which is also an alias I use in my files and suggest for common use.

When using the "dsl_usings.h" header, the following keywords are added:

1. dsl – shorter way to refer to the library functions.
2. output – this is from dsl::file_type::output
3. input – this is from dsl::file_type::input
4. console – this is from dsl::file_type::console

1.1 Introduction

I found it a bit of a pain to code up many of the ubiquitous algorithms in combinatorics and probability. In particular, in combinatorial probability one often wishes to generate all objects of a given weight for small weights, or randomly sample objects of a large weight, in order to search for macroscopic properties.

Example: Counting. the log-concavity of the integer partition function is something which can be observed for the first billion or so values, excluding the first 25 (only works for even values). One should like to exhaust all small cases first before attempting to prove such a result; see `http://link.springer.com/article/10.-1007%2Fs11139-014-9599-y#page-1` or `http://arxiv.org/abs/1310.7982`.

Example: Patterns. Suppose we wish to explore bijections between various combinatorial objects, e.g., integer partitions into distinct parts vs integer partitions into odd parts. We would first like to be able to count them to make sure that there are indeed the same number of objects in each set. Then, to search for a combinatorial bijection, it would be preferable to generate a comprehensive list for small weights and attempt to find a pattern, which can be proven to continue for all n.

Example: Random sampling. For combinatorial objects with larger weight, a complete enumeration is infeasible, and so we often wish to sample, uniformly at random, such an object. Efficient random samplers are often difficult to obtain; even if one has a polynomial time algorithm, a O(n^2) vs O(n) time algorithm makes a large difference in practice, as well as the memory constraints.

Conceptual Solution: Integer partitions are one example of a combinatorial structure for which one might wish to look at the number of such objects, list them, list a subset, randomly generate a subset, etc. It turns out that they are a typical example of a "decomposable" combinatorial structure, see http://arxiv.org/abs/1308.3279, and one can actually treat the other such structures in generality.

In fact, the code that one writes is essentially the same in all cases, hence this library was designed to attempt to minimize the amount of duplicate code one would need to write in order to facilitate the above tasks as easily as possible.

C++ Coding Solution: Our main tool is curiously recurring template pattern (CRTP), which allows us to write code in a base class, templated so that it calls the functions of the inherited class. The effect is similar to much of mathematical theory, e.g., vector spaces: one starts by defining a list of rules that apply to all objects in a vector space X over a field F. As long as F and X satisfy certain properties, then all theorems assuming that X is a vector space apply. In this case, X acts like a base class, which we have provided, and F is the class specified by the programmer. As long as F satisfies certain properties, then X will utilize those properties, from which others can be derived.

1.2 Overview of libraries

The libraries are divided into several categories.

1. Fundamental
2. CRTP base class
3. CRTP derived class
4. Utility

1.2.1 Fundamental

These are files like numerical.h, statistics.h, file.h. These classes provide basic functionality which should be part of any general-purpose mathematical library of functionality.

1.2.2 CRTP base class

These are files like sequence.h, which provide a core set of functionality so that when a class is derived from them, they obtain all of the desired functionality.

1.2.3 CRTP derived class

These are files like permutation.h, which contains classes for permutations which inherit from a CRTP base class and provide only a few necessary functions in order to unlock functionality like complete enumeration or random generation.

1.2.4 Utility

Finally, there are files like std_cout.h, dsl_algorithm.h, dsl.h, dsl_usings.h, which are mostly wrappers around existing functionality. They are meant to make working with collections of objects easier, for example allowing for Matlab-like syntax in a C++ environment.

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all documented namespaces 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 documented files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 desalvo_standard_library Namespace Reference

think of this namespace like std or boost, I typically use dsl as an alias.

Classes

- class binary_integer

  Stores an binary_integer value using bits.
- class north_east_lattice_path

  all walks from (0,0) to (n,k) using up and right moves
- class binary_contingency_table
- class binary_contingency_table_set
- class binary_contingency_table_set< EntryType, SumType, VectorSumType, store::bidirectional, Probability-TableType >
- class binary_contingency_table_generator
- class file

  Partially specialized for input and output.
- class file< file_type::input >
- class file< file_type::output >
- class file< file_type::console >
- class Fraction

  Fraction class for storing int/int with arithmetic operators defined.
- class integer_partition_generator
- class integer_partition
- class decomposable_structure
- class latin_square
- class matrix
- class NotDivisibleBy

  Creates function objects which check for divisibility.
- class DivisibleBy

  Creates function objects which check for divisibility.
- class ArithmeticProgression

  Sequence generator for an arithmetic progression {a, a+r, a+2r, ...}.
- class numerical_table
- class permutation
- class permutation< dsl::store::random_access, restrictions::fixed_point_free, T, V, SV >
- class permutation< dsl::store::bidirectional, restrictions::fixed_point_free, T, V, SV >

- class permutation< dsl::store::forward, restrictions::fixed_point_free, T, V, SV >
- class permutation< dsl::store::random_access, restrictions::none, T, V, SV >
- class permutation< dsl::store::bidirectional, restrictions::none, T, V, SV >
- class permutation< dsl::store::forward, restrictions::none, T, V, SV >
- class permutation< dsl::store::random_access, restrictions::by_pairs, T, V, SV >
- class permutation< dsl::store::bidirectional, restrictions::by_pairs, T, V, SV >
- class permutation< dsl::store::forward, restrictions::by_pairs, T, V, SV >
- class permutation< dsl::store::random_access, restrictions::by_function, T, V, SV >
- class permutation< dsl::store::bidirectional, restrictions::by_function, T, V, SV >
- class permutation< dsl::store::forward, restrictions::by_function, T, V, SV >
- class sequence_parameters
- class finite_sequence

  A CRTP class for working with finite sequences.
- class finite_sequence< dsl::store::random_access, Derived, T, V >
- class finite_sequence< dsl::store::bidirectional, Derived, T, V >
- class finite_sequence< dsl::store::forward, Derived, T, V >
- class finite_sequence_threadable

  A CRTP class for working with finite sequences.
- class finite_sequence_threadable< dsl::store::forward, Derived, T, V >
- class set_partition
- class shrinking_set_unordered

  initialized with a set of objects, then efficiently erases and resets again
- class shrinking_set

  initialized with a set of objects, then efficiently erases and resets again keeping non-erased elements in sorted order
- class random_variable
- class discrete_uniform
- class real_uniform
- class truncated_geometric_distribution

  truncated geometric distribution
- class random_distinct_subset

  Generates a subset of size k from {1,2,...,n}.
- class numeric_data

  stores collections of numeric data, calculates statistics
- class sudoku
- class table

  stores a 2-dimensional table of values
- class time

  A class for keeping track of timings easily.

Typedefs

- typedef std::ostream &(∗ manip1)(std::ostream &)

  abbrev. for type 1 manipulators
- typedef std::basic_ios
  < std::ostream::char_type,
  std::ostream::traits_type > ios_type

  abbrev. for use with typedef for type 2 manipulators
- typedef ios_type &(∗ manip2)(ios_type &)

  abbrev. for type 2 manipulators
- typedef std::ios_base &(∗ manip3)(std::ios_base &)

  abbrev. for type 3 manipulators
- typedef unsigned long long **ull**

Enumerations

- enum file_type { **input**, **output**, **console** }
- enum **IPARCS** { **ASSEMBLY**, **MULTISET**, **SELECTION** }
- enum restrictions { **none**, **fixed_point_free**, **by_pairs**, **by_function** }

  either Unrestricted, or Restrictions listed by either pairs of (i, sigma(i)), or by a binary matrix.
- enum store { **random_access**, **bidirectional**, **forward** }
- enum **SimulationMethod** { **BruteForce**, **Boltzmann**, **BoltzmannExact**, **PDCDSH** }

Functions

- bool operator!= (const binary_integer &lhs, const binary_integer &rhs)
- bool operator> (const binary_integer &lhs, const binary_integer &rhs)
- bool operator<= (const binary_integer &lhs, const binary_integer &rhs)
- bool operator>= (const binary_integer &lhs, const binary_integer &rhs)
- binary_integer operator+ (binary_integer lhs, const binary_integer &rhs)
- binary_integer operator- (binary_integer lhs, const binary_integer &rhs)
- binary_integer operator* (binary_integer lhs, const binary_integer &rhs)
- binary_integer operator& (binary_integer lhs, const binary_integer &rhs)
- binary_integer operator| (binary_integer lhs, const binary_integer &rhs)
- binary_integer operator^ (binary_integer lhs, const binary_integer &rhs)
- binary_integer operator<< (binary_integer lhs, const binary_integer &rhs)
- binary_integer operator>> (binary_integer lhs, const binary_integer &rhs)
- int **char_to_int** (char c)
- char **digit_to_char** (int a)
- std::ostream & **operator**<< (std::ostream &out, const binary_integer::binary_integer_string &a)
- std::ostream & **operator**<< (std::ostream &out, const binary_integer &a)
- std::istream & **operator**>> (std::istream &in, binary_integer &a)
- template<typename V >

  void iota (V &v, typename V::value_type val=static_cast< typename V::value_type >(1))
- template<typename V >

  bool next_permutation (V &v)
- template<typename V , typename Compare >

  bool next_permutation (V &v, Compare &&cmp)
- template<typename V >

  bool prev_permutation (V &v)
- template<typename V , typename Compare >

  bool prev_permutation (V &v, Compare cmp)
- bool getline (file< file_type::input > &fin, std::string &s)
- template<typename String >

  bool getline (file< file_type::console > &fin, String &s)
- template<typename T >

  Fraction< T > operator+ (Fraction< T > f, const Fraction< T > &f2)
- template<typename T >

  Fraction< T > operator- (Fraction< T > f, const Fraction< T > &f2)
- template<typename T >

  Fraction< T > operator* (Fraction< T > f, const Fraction< T > &f2)
- template<typename T >

  Fraction< T > operator/ (Fraction< T > f, const Fraction< T > &f2)
- template<typename T >

  bool operator== (const Fraction< T > &lhs, const Fraction< T > &rhs)
- template<typename T >

  bool operator< (const Fraction< T > &lhs, const Fraction< T > &rhs)
- template<typename T >

  bool operator!= (const Fraction< T > &lhs, const Fraction< T > &rhs)

- template<typename T >
  bool operator> (const Fraction< T > &lhs, const Fraction< T > &rhs)
- template<typename T >
  bool operator<= (const Fraction< T > &lhs, const Fraction< T > &rhs)
- template<typename T >
  bool operator>= (const Fraction< T > &lhs, const Fraction< T > &rhs)
- template<typename T >
  T gcd (T a, T b)
- template<typename Integer >
  Integer **Euler** (const Integer &n, const Integer &m)
- template<typename ValueType = unsigned int, typename WorkingPrecision = unsigned long int, typename Parameters = std::vector<int>>
  latin_square< ValueType,
  WorkingPrecision >::template
  object< Parameters > **random_latin_square** (ValueType n)
- template<typename ValueType = double, typename WorkingPrecision = long double>
  bool **operator!=** (const matrix< ValueType, WorkingPrecision > &lhs, const matrix< ValueType, WorkingPrecision > &rhs)
- template<typename ValueType = double, typename WorkingPrecision = long double>
  matrix< ValueType,
  WorkingPrecision > **identity_matrix** (size_t n)
- template<typename ValueType = double, typename WorkingPrecision = long double>
  matrix< ValueType,
  WorkingPrecision > **all_ones** (size_t m, size_t n)
- template<typename Integer , typename UnsignedInteger = Integer>
  UnsignedInteger gcd (Integer a, Integer b)
- template<typename F = double, typename V = std::vector<F>, typename Size = size_t>
  V range (Size n, F initial_value=1.)
- template<typename F = double, typename V = std::vector<F>, typename Size = size_t>
  V constant_array (Size n, F initial_value=1.)
- template<typename Size = size_t, typename V = std::vector<std::pair<Size,Size>>>
  V table_indices (Size m, Size n, Size initial_value_first=0, Size initial_value_second=0)
- template<typename V , typename Comparison = std::less<typename V::value_type>>
  void sort_in_place (V &v, Comparison cmp=std::less< typename V::value_type >())
- template<typename F = double, typename V = std::vector<F>>
  void partial_sum_in_place (V &v)
- template<typename T = bool, typename Vector = std::vector<T>>
  Vector binary_row (size_t n, size_t k, T val=true)
- template<typename V >
  void reverse_in_place (V &v)
- template<typename T , typename F = T>
  F factorial (T n)
- template<typename T1 , typename T2 , typename F = T1>
  F nfallingk (T1 n, T2 k)
- template<typename T1 , typename T2 , typename F = T1>
  F binomial (T1 n, T2 k)
- template<typename T , typename F = T>
  F choose2 (T n)
- template<typename T , typename F = T>
  F choose3 (T n)
- template<typename T , typename F = T>
  F choose4 (T n)
- template<typename N , typename T , typename F = T>
  F binomial_probability (N n, N k, T p)
- template<typename V , typename C >
  void print_side_by_side (const V &left, const C &right, const std::string &sep=std::string(" "), const std::string &endline=std::string("\n"))

- template<typename InputIterator1 , typename InputIterator2 >
  void print_side_by_side (InputIterator1 start1, InputIterator1 stop, InputIterator2 start2, const std::string &sep=std::string(" "), const std::string &endline=std::string("\n"))
- template<typename ReturnValueType = double, typename IntegerType = long long int, typename DataType = ReturnValueType, typename InputIterator = typename std::vector<DataType>::iterator>
  ReturnValueType sum_of_powers (InputIterator start, InputIterator stop, IntegerType power, DataType initial=0.)
- template<typename T >
  std::vector< std::vector< T > > permutations (std::vector< T > objects)
- template<typename IntegerType , typename ContainerType = std::vector<IntegerType>>
  ContainerType int_to_digits (IntegerType a, bool left_to_right=true)
- template<typename IntegerType = int, typename ContainerType = std::vector<IntegerType>>
  IntegerType digits_to_int (ContainerType digits, bool is_left_to_right=true)
- template<typename Iterator , typename IntegerType >
  bool is_permutation_of_n (Iterator start, const Iterator &stop, IntegerType n)
- template<typename Container , typename BinaryPredicate = std::equal_to<typename Container::value_type>, typename Comparison = std::less<typename Container::value_type>>
  bool has_unique_elements (Container elements, BinaryPredicate pred=std::equal_to< typename Container::value_type >(), Comparison cmp=std::less< typename Container::value_type >())
- template<typename UnsignedIntegers >
  bool is_unique_uints_max_31 (UnsignedIntegers values)
- template<typename ForwardIterator >
  bool is_unique_uints_max_31 (ForwardIterator first, ForwardIterator last)
- template<typename V >
  V conjugate_integer_partition (V v)
- template<typename T , typename RandomAccess >
  void sort_between (RandomAccess start, RandomAccess stop, T val)
- template<typename T , typename ForwardIterator >
  ForwardIterator binary_search_iterator (ForwardIterator start, ForwardIterator stop, T &&t)
- template<typename T , typename ForwardIterator >
  ForwardIterator binary_search_iterator_first (ForwardIterator start, ForwardIterator stop, T &&t)
- template<typename _InputIterator , typename Size , typename _OutputIterator , typename _UnaryOperation >
  void transform_n (_InputIterator __first, Size __n, _OutputIterator __result, _UnaryOperation __op)
- template<typename _InputIterator1 , typename Size , typename _InputIterator2 , typename _OutputIterator , typename _BinaryOperation >
  void transform_n (_InputIterator1 __first1, Size __n, _InputIterator2 __first2, _OutputIterator __result, _BinaryOperation __binary_op)
- template<typename InputIterator , typename OutputIterator >
  OutputIterator unique_copy_nonconsecutive (InputIterator start, InputIterator stop, OutputIterator output)
- template<typename InputIterator , typename OutputIterator , typename BinaryPredicate >
  OutputIterator unique_copy_nonconsecutive (InputIterator start, InputIterator stop, OutputIterator output, BinaryPredicate bin_op)
- template<class RandomAccessIterator >
  void transpose (RandomAccessIterator first, RandomAccessIterator last, size_t m)
- template<typename V = std::vector<size_t>>
  V sieve (size_t n)
- std::vector< std::vector< short > > multiset_subsets (short n, short k)
- std::vector< std::vector< short > > unique_multiset_subsets (short n, short k)
- size_t two_by_two_map (const std::vector< short > &v, const std::vector< std::vector< short >> &possibles)
- size_t **two_by_two_map** (const std::vector< short > &v, const std::vector< std::pair< std::vector< short >, double >> &possibles)
- std::vector< unsigned int > fizz_buzz_partition (size_t n)
- template<typename IntType = size_t, typename Container = std::vector<std::vector<IntType>>>
  Container permutation_as_product_of_cycles (const std::vector< IntType > &permutation)
- template<typename IntType , typename Container = std::vector<std::vector<IntType>>>
  Container permutation_as_product_of_transpositions (const std::vector< IntType > &permutation)

- std::vector< size_t > permutation_reduction (std::vector< size_t > vals)
- template<typename ValueType = double, typename WorkingPrecision = long double>
  numerical_table< ValueType,
  WorkingPrecision > **operator+** (numerical_table< ValueType, WorkingPrecision > lhs, const numerical_table< ValueType, WorkingPrecision > &rhs)
- template<typename T , typename ValueType = double, typename WorkingPrecision = long double>
  numerical_table< ValueType,
  WorkingPrecision > **operator+** (numerical_table< ValueType, WorkingPrecision > lhs, const T &value)
- template<typename T , typename ValueType = double, typename WorkingPrecision = long double>
  numerical_table< ValueType,
  WorkingPrecision > **operator+** (const T &value, numerical_table< ValueType, WorkingPrecision > lhs)
- template<typename ValueType = double, typename WorkingPrecision = long double>
  numerical_table< ValueType,
  WorkingPrecision > **operator-** (numerical_table< ValueType, WorkingPrecision > lhs, const numerical_table< ValueType, WorkingPrecision > &rhs)
- template<typename T , typename ValueType = double, typename WorkingPrecision = long double>
  numerical_table< ValueType,
  WorkingPrecision > **operator-** (numerical_table< ValueType, WorkingPrecision > lhs, const T &value)
- template<typename T , typename ValueType = double, typename WorkingPrecision = long double>
  numerical_table< ValueType,
  WorkingPrecision > **operator-** (const T &value, numerical_table< ValueType, WorkingPrecision > lhs)
- **finite_threadable** (input_n)
- template<typename V = std::vector<int>, typename InputIterator = std::vector<int>::iterator>
  V **reduction** (InputIterator start, InputIterator stop)
- template<typename ValueType = unsigned int, typename WorkingPrecision = unsigned long int, typename Parameters = std::vector<int>>
  set_partition< ValueType,
  WorkingPrecision >::template
  object< Parameters > **random_set_partition** (ValueType n)
- template<typename T >
  T random_integer (T a, T b)
- template<typename T , typename V = std::vector<T>>
  V random_integer_vector (T a, T b, size_t n)
- template<typename N = std::size_t, typename V = std::vector<N>, typename URNG = std::mt19937_64>
  V random_permutation (N n, URNG &gen=generator_64)
- template<typename T = bool, typename Vector = std::vector<T>, typename URNG = std::mt19937_64>
  Vector random_binary_row (size_t n, size_t k, T val=true, URNG &gen=generator_64)
- template<typename N = size_t, typename Float = long double, typename V = std::vector<N>, typename URNG = std::mt19937_64>
  V random_permutation_mallows_in_mallows_form (N n, Float q, URNG &gen=generator_64)
- template<typename N = size_t, typename Float = long double, typename URNG = std::mt19937_64>
  std::vector< N > random_permutation_mallows_ordering_construction (N n, Float q, URNG &gen=generator_64)
- template<typename N = size_t, typename Float = long double, typename V = std::vector<N>, typename URNG = std::mt19937_64>
  V random_permutation_mallows (N n, Float q, URNG &gen=generator_64)
- template<typename N = std::size_t, typename V = std::vector<N>, typename URNG = std::mt19937_64>
  V random_permutation_shifted (N n, N a, URNG &gen=generator_64)
- template<typename N = std::size_t, typename V = std::vector<N>, typename URNG = std::mt19937_64>
  V random_permutation_fixed_point_free (N n, URNG &gen=generator_64)
- template<typename URNG = std::mt19937>
  size_t uniform_size_t (size_t a, size_t b, URNG &gen=generator_32)
- template<typename T = bool, typename Vector = std::vector<T>, typename N = size_t, typename URNG = std::mt19937_64>
  Vector bernoull_iid_fixedsum (N n, N k, URNG &gen=generator_64)
- template<typename T = bool, typename Vector = std::vector<T>, typename N = size_t, typename URNG = std::mt19937_64>
  Vector set_n_choose_k (N n, N k, URNG &gen=generator_64)
- template<typename T = bool, typename Vector = std::vector<T>, typename N = size_t, typename URNG = std::mt19937_64>
  Vector set_2n_choose_n (N n, URNG &gen=generator_64)

- template<typename N = size_t, typename V = std::vector<N>, typename URNG = std::mt19937_64>
  V partial_permutation_rejection (N n, N k, URNG &gen=generator_64)
- template<typename N = size_t, typename V = std::vector<N>, typename URNG = std::mt19937_64>
  V partial_permutation (N n, N k, URNG &gen=generator_64)
- template<typename T = bool, typename Vector = std::vector<T>, typename N = size_t, typename Real = double, typename URNG = std::mt19937_64>
  Vector **set_n_choose_k_repeated** (N n, N k, URNG &gen=generator_64)
- template<typename V = std::vector<bool>, typename T = std::vector<double>, typename N = size_t, typename URNG = std::mt19937_64>
  V bernoulli_fixedsum_rejection (const T &p, N k, URNG &gen=generator_64)
- template<typename V = std::vector<bool>, typename T = std::valarray<double>, typename N = size_t, typename URNG = std::mt19937_64, typename F = double>
  V poisson_fixedsum_poisson_process (const T &p, N k, URNG &generator=generator_64)
- template<typename T , typename String = std::string>
  void read (T &container, std::istream &in=std::cin)
- template<typename T , typename String = std::string>
  void print (T &&container, std::string ending="", std::ostream &out=std::cout, String separation=std::string(","), String open_bracket=std::string("{"), String close_bracket=std::string("}"))
- template<typename T , typename String = std::string>
  void **print** (T &&container, std::string begin_with="{", std::string separate_by=",", std::string end_with="}", std::ostream &out=std::cout)
- template<typename ValueType , typename WorkingPrecision >
  bool **operator!=** (const table< ValueType, WorkingPrecision > &lhs, const table< ValueType, WorkingPrecision > &rhs)

Variables

- **last_element** = last_in_sequence()

6.1.1 Detailed Description

think of this namespace like std or boost, I typically use dsl as an alias. The core set of functionality is contained in this namespace. It consists of the content in the following files:

1. numerical.h
2. statistics.h
3. std_cout.h
4. file.h
5. shrinking_set.h
6. sequence.h
7. permutation.h
8. dsl_algorithm.h

Templated function for output of std::multiset< std::pair<size_t, size_t> >::iterator format. Not sure why I made this function ...

Parameters

out	is the output stream
my_list	is the list<T> to output

Returns

the output stream.

6.1.2 Enumeration Type Documentation

6.1.2.1 enum desalvo_standard_library::file_type

controls the type of file

6.1.2.2 enum desalvo_standard_library::restrictions

either Unrestricted, or Restrictions listed by either pairs of (i, sigma(i)), or by a binary matrix.

The choices at present are: {none, by_pairs, by_matrix}

6.1.2.3 enum desalvo_standard_library::store

is the internal storage

6.1.3 Function Documentation

6.1.3.1 template<typename T = bool, typename Vector = std::vector<T>, typename N = size_t, typename URNG = std::mt19937_64> Vector desalvo_standard_library::bernoulli_iid_fixedsum (N *n*, N *k*, URNG & *gen* = `generator_64`)

Randomly sample iid Bernoulli's conditional on having sum k. O(n) O(n log n) due to the shuffle operation

Parameters

n	is the number of Bernoulli random variables
k	is the number of 1s.

Template Parameters

gen	is the random generator, by default 64-bit

Returns

an ordered vector of n 0s and 1s, exactly k of which are 1s.

6.1.3.2 template<typename V = std::vector<bool>, typename T = std::vector<double>, typename N = size_t, typename URNG = std::mt19937_64> V desalvo_standard_library::bernoulli_fixedsum_rejection (const T & *p*, N *k*, URNG & *gen* = `generator_64`)

Randomly sample Bernoulli's with different parameters conditional on having sum k.

Parameters

p	is a vector of probabilities
k	is the number of 1s.

Template Parameters

gen	is the random generator, by default 64-bit

Returns

a vector of n 0s and 1s, exactly k of which are 1s in some random locations.

6.1.3.3 template<typename T = bool, typename Vector = std::vector<T>> Vector desalvo_standard_library::binary_row (size_t *n*, size_t *k*, T *val* = `true`)

Creates a new container with the partial sums, needs begin and end defined

Template Parameters

F	is the value type
V	is the container type

Parameters

v	is the container to sort elements in place. Returns a vector (val,val,...,val,[0],[0],...,[0]) of k vals and n-k [0]s, where [0] is the default value of the container.
n	is the size of the vector
k	is the number of vals

Template Parameters

val	is the value to fill in

Returns

(val,val,...,val,[0],[0],...,[0]) of k vals and n-k [0]s

```
#include "desalvo/numerical.h" // See documentation for list of keywords included in
      this file.
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

auto v = dsl::binary_row(10, 3);
auto v2 = dsl::binary_row(10, 3,4);
auto v3 = dsl::binary_row(10, 3,3.14);

std::cout << v << std::endl;
std::cout << v2 << std::endl;
std::cout << v3 << std::endl;

return 0;
}
```

Should produce output:

```
{1,1,1,0,0,0,0,0,0,0}
{4,4,4,0,0,0,0,0,0,0}
{3.14,3.14,3.14,0,0,0,0,0,0,0}
```

6.1.3.4 template<typename T , typename ForwardIterator > ForwardIterator desalvo_standard_library::binary_search_iterator (ForwardIterator *start*, ForwardIterator *stop*, T && *t*)

Intention is to deprecate since std::lower_bound is the intended functionality: Returns an iterator indexed by the forward iterators to the value input, or an iterator equivalent to stop.

Template Parameters

T	is the type of the value to search for
ForwardIterator	is any forward iterator type

Parameters

start	is the beginning of the collection
stop	is one after the last element

Returns

iterator to found element, else iterator equivalent to stop

6.1.3.5 template<typename T , typename ForwardIterator > ForwardIterator desalvo_standard_library::binary_search_iterator_first (ForwardIterator *start*, ForwardIterator *stop*, T && *t*)

Returns an iterator using the first of a pair of elements indexed by the random access iterators.

Template Parameters

T	is the type of the value to search for
ForwardIterator	is any forward iterator type

Parameters

start	is the beginning of the collection
stop	is one after the last element

Returns

iterator to found element, else iterator equivalent to stop

6.1.3.6 template<typename T1 , typename T2 , typename F = T1> F desalvo_standard_library::binomial (T1 *n*, T2 *k*)

calculates the binomial coefficient

Template Parameters

T	is the input type
F	is the output type

Parameters

n	is the larger value
k	is the smaller value

Returns

n choose k

Example 1:

```
#include "desalvo/numerical.h" // See documentation for list of keywords included in
```

```
        this file.
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

// typdef for large unsigned integer
typedef unsigned long long ull;

int main(int argc, const char * argv[]) {

// We must be careful!  By default we get a signed integer type, which is not defined in cases of overflow
auto x = dsl::binomial(50,10);
std::cout << x << std::endl; // 106   <-- overflow!

// the function is templated so that you can specify the data type for which the calculations will be
        performed, in this case an unsigned long long is large enough.
auto x2 = dsl::binomial<ull>(50,10);
std::cout << x2 << std::endl;   // 10272278170   <-- Ok!

// Make sure you know what you are doing!  I get a run-time error since binomial calls
        nfallingk(n,k)/factorial(k), and in this case, nfallingk(n,k) has overflow and returns 0, and factorial(k) has overflow
        returns 0.  The reason is because ULLONG_MAX is a multiple of 2, typically 2^64, and so after 64 factors of 2
        have been multiplied together, we obtain a multiple of 2^64, hence taking modulo 2^64 gives 0.
// So the error is in attempting to compute 0/0.  Ask Siri if you don't understand the implications.
auto x3 = dsl::binomial<ull>(1000,100);
std::cout << x3 << std::endl;   // Run-time error at this point for me.

return 0;
}
```

Should produce output (depending on numeric limits)

```
106
10272278170
(run-time error occurs)
```

Example 2:

```
#include "desalvo/dsl_usings.h" // See documentation for list of keywords included in
        this file.

int main(int argc, const char * argv[]) {
   std::cout << dsl::binomial(10,-2) << std::endl;
   std::cout << dsl::binomial(0,-2) << std::endl;
   std::cout << dsl::binomial(1,0) << std::endl;
   std::cout << dsl::binomial(0,1) << std::endl;
   std::cout << dsl::binomial(0,2) << std::endl;
   std::cout << dsl::binomial(0,-10) << std::endl;
   std::cout << dsl::binomial(-5,-2) << std::endl;

   return 0;
}
```

Should produce output

```
0
0
1
0
0
0
0
0
```

6.1.3.7 template<typename N , typename T , typename F = T> F desalvo_standard_library::binomial_probability (N *n*, N *k*, T *p*)

Calculates the binomial probability (n choose k) p^k (1-p)^(n-k) in a way as numerically stable as I could make it for now.

Parameters

n	is the number of trials
k	is the number of successes
p	is the probability of success

Template Parameters

N	is the integer type
T	is the floating point type
F	is the return type

Returns

the probability of k successes in n trials with probability of success p.

Example 1:

```
#include "desalvo/dsl_usings.h" // See documentation for list of keywords included in
       this file.

int main(int argc, const char * argv[]) {

for(int i=0;i<=100;++i)
std::cout << dsl::binomial_probability(100,i,0.5) << std::endl;

return 0;
}
```

6.1.3.8 template<typename T , typename F = T> F desalvo_standard_library::choose2 (T *n*)

calculates n(n-1)/2, a common binomial coefficient

Template Parameters

T	is the input type
F	is the output type

Parameters

n	is the number of elements to choose 2 from

Returns

n choose 2

```
#include "desalvo/numerical.h" // See documentation for list of keywords included in
       this file.
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

// typdef for large unsigned integer
typedef unsigned long long ull;

int main(int argc, const char * argv[]) {

// We must be careful!  By default we get a signed integer type, which is not defined in cases of overflow
auto x = dsl::choose2(1234567);
std::cout << x << std::endl; // -279473579   <-- overflow!

// the function is templated so that you can specify the data type for which the calculations will be
       performed, in this case an unsigned long long is large enough.
auto x2 = dsl::choose2<ull>(1234567);
std::cout << x2 << std::endl;   // 762077221461   <-- Ok!

// Make sure you know what you are doing!
auto x3 = dsl::choose2<ull>(1234567891011);
std::cout << x3 << std::endl;   // 7047583967906995363  <-- modulo 2^64

// Pop quiz:  Is this the right answer?
std::cout << sqrt(ULLONG_MAX) << std::endl;  // 4.29497e+09   <-- take the n choose 2 of this
auto x4 = dsl::choose2<ull>( sqrt(ULLONG_MAX) );
std::cout << x4 << std::endl;   // 9223372034707292160  <-- Is this the actual answer?
```

```
return 0;
}
```

Should produce output (depending on numeric limits)

```
-279473579
762077221461
7047583967906995363
4.29497e+09
9223372034707292160
```

6.1.3.9 template<typename T , typename F = T> F desalvo_standard_library::choose3 (T *n*)

calculates n(n-1)(n-2)/3, a common binomial coefficient

Template Parameters

T	is the input type
F	is the output type

Parameters

n	is the number of elements to choose 3 from

Returns

n choose 3

```
#include "desalvo/numerical.h" // See documentation for list of keywords included in
       this file.
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

// typdef for large unsigned integer
typedef unsigned long long ull;

int main(int argc, const char * argv[]) {

// We must be careful!  By default we get a signed integer type, which is not defined in cases of overflow
auto x = dsl::choose3(1234567);
std::cout << x << std::endl; // -230422855   <-- overflow!

// the function is templated so that you can specify the data type for which the calculations will be
       performed, in this case an unsigned long long is large enough.
auto x2 = dsl::choose3<ull>(1234567);
std::cout << x2 << std::endl;   // 313611288304333155   <-- Ok!

// Make sure you know what you are doing!
auto x3 = dsl::choose3<ull>(123456789);
std::cout << x3 << std::endl;   // 2699470946007441500  <-- n choose 3 modulo 2^64

return 0;
}
```

Should produce output (depending on numerical limits)

```
-230422855
313611288304333155
2699470946007441500
```

6.1.3.10 template<typename T , typename F = T> F desalvo_standard_library::choose4 (T *n*)

calculates n choose 4, a common binomial coefficient

Template Parameters

T	is the input type
F	is the output type

Parameters

n	is the number of elements to choose 4 from

Returns

n choose 4

```
#include "desalvo/numerical.h" // See documentation for list of keywords included in
      this file.
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

// typdef for large unsigned integer
typedef unsigned long long ull;

int main(int argc, const char * argv[]) {

// We must be careful!  By default we get a signed integer type, which is not defined in cases of overflow
auto x = dsl::choose4(12345);
std::cout << x << std::endl; // -69762984   <-- overflow!

// the function is templated so that you can specify the data type for which the calculations will be
      performed, in this case an unsigned long long is large enough.
auto x2 = dsl::choose4<ull>(12345);
std::cout << x2 << std::endl;   // 967257345895170   <-- Ok!

// Make sure you know what you are doing!
auto x3 = dsl::choose4<ull>(123456789);
std::cout << x3 << std::endl;   // 98740589912719051  <-- n choose 4 modulo 2^64

return 0;
}
```

Should produce output (depending on numerical limits)

```
-69762984
967257345895170
98740589912719051
```

6.1.3.11 template<typename V > V desalvo_standard_library::conjugate_integer_partition (V *v*)

Finds the conjugate partition for a given input, entries can be in any order. The container needs RANDOM ACCESS iterators in order to guarantee that the entries are in sorted order. Otherwise bidirectional is sufficient if already sorted.

Template Parameters

F	is the value type
V	is the container type

Parameters

v	is an integer vector of values

Returns

the conjugate partition in descending order

```
#include "desalvo/std_cout.h"
```

```
#include "desalvo/numerical.h"

namespace dsl = desalvo_standard_library;


int main(int argc, const char * argv[]) {

std::vector<int> v {1,1,1,2,2,3,4,5,5,5,6,6,8,9,10,11};
std::vector<int> v2 {1,1,2,4,7};
std::vector<int> v3 {1,10,5};

auto s = dsl::conjugate_integer_partition(v);
auto s2 = dsl::conjugate_integer_partition(v2);
auto s3 = dsl::conjugate_integer_partition(v3);


dsl::print(s, "\n");
dsl::print(s2, "\n");
dsl::print(s3, "\n");

return 0;
}
```

Should produce output

```
(not working at the moment)
```

6.1.3.12 template<typename F = double, typename V = std::vector<F>, typename Size = size_t> V desalvo_standard_library::constant_array (Size *n*, F *initial_value*)

Initializes list to {initial_value, initial_value+1,...,initial_value+n-1}. By default, the initial value of 1.

Template Parameters

Size	is any nonnegative integer type
V	is the container to store the values
F	is the data type of the values

Parameters

n	is the size of the collection
initial_value	is the value of the first element.

Returns

a collection of values starting with initial_value and adding 1 n-1 times

```
// Example of using dsl::range
#include "desalvo/numerical.h" // See documentation for list of keywords included in
      this file.
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// range calls the += operator, so can be used with any numerical container as well like valarray
auto v = dsl::range(10);
auto v2 = dsl::range(10, 0.1);
auto v3 = dsl::range(10, 0);
auto v4 = dsl::range(10, std::valarray<short>({0,1,2,3,4}));

std::cout << v << std::endl;
std::cout << v2 << std::endl;
std::cout << v3 << std::endl;
std::cout << v4 << std::endl;

return 0;
}
```

6.1.3.13 template<typename IntegerType = int, typename ContainerType = std::vector<IntegerType>> IntegerType desalvo_standard_library::digits_to_int (ContainerType *digits*, bool *is_left_to_right* = true)

Converts the digits in the input container into a single number

Template Parameters

IntegerType	is any unsigned integer type
ContainerType	is the container to store digits

Parameters

digits	is a collection of digits of a stored individually

Returns

a numerical value for which the digits in digits were representing (base 10)

```
#include "desalvo/std_cout.h"
#include "desalvo/numerical.h" // See documentation for list of keywords included in
       this file.

namespace dsl = desalvo_standard_library;


int main(int argc, const char * argv[]) {

// converts digits into a single value base 10.
std::vector<int> v {3,1,4,1,5,9,2,6,5};
auto x = dsl::digits_to_int(v);
dsl::print(x,"\n");

// There is no check for overflow, but ...
std::vector<int> v2 {1,2,3,4,5,6,7,8,9,1,0,1,1,1};
auto x2 = dsl::digits_to_int(v2);
dsl::print(x2,"\n");

// You can at least be a little smart about it and give a larger container
std::vector<int> v3 {1,2,3,4,5,6,7,8,9,1,0,1,1,1};
auto x3 = dsl::digits_to_int<unsigned long long>(v3);
dsl::print(x3,"\n");

// Reverse, reverse!
x3 = dsl::digits_to_int<unsigned long long>(v3, false);
dsl::print(x3,"\n");


return 0;
}
```

Should produce output (depending on numerical limits)

```
314159265
1942901407
12345678910111
11101987654321
```

6.1.3.14 template<typename T , typename F = T> F desalvo_standard_library::factorial (T *n*)

Calculates the factorial on input of type T and outputs in type F

Template Parameters

T	is the input integer type
F	is the output integer type

Parameters

n	is the value for which to take the factorial

Returns

n! in data type F

```
#include "desalvo/numerical.h" // See documentation for list of keywords included in
       this file.
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

// typdef for large unsigned integer
typedef unsigned long long ull;

int main(int argc, const char * argv[]) {

// We must be careful!  By default we get a signed integer type, which is not defined in cases of overflow
auto x = dsl::factorial(20);
std::cout << x << std::endl; // -2102132736   <-- overflow!

// the function is templated so that you can specify the data type for which the calculations will be
       performed, in this case an unsigned long long is large enough.
auto x2 = dsl::factorial<ull>(20);
std::cout << x2 << std::endl;   // 2432902008176640000   <-- Ok!

// Make sure you know what you are doing!  This answer returned is actually 30! % ULLONG_MAX.  This
       behavior is in fact well-defined for unsigned types in C++, but you should always be careful when doing
       calculations involving extremely fast-growing numerical values.
auto x3 = dsl::factorial<ull>(30);
std::cout << x3 << std::endl;   // 9682165104862298112  <-- 30! % ULLONG_MAX

return 0;
}
```

Should produce output (depending on numeric limits)

```
-2102132736
2432902008176640000
9682165104862298112
```

6.1.3.15 std::vector<unsigned int> desalvo_standard_library::fizz_buzz_partition (size_t *n*)

Partitions the first n numbers {1,2,...,n} into (not divisible by 3 nor 5 | divisible by just 3 | divisible by just 5 | divisible by both 3 and 5)

Parameters

n — is the size of the list, for numbers {1,2,...,n}

```
#include "desalvo/numerical.h"
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

dsl::print(dsl::fizz_buzz_partition(25));

return 0;
}
```

Should produce output

```
{1,2,4,7,8,11,13,14,16,17,19,22,23,3,6,9,12,18,21,24,5,10,20,25,15}
```

6.1.3.16 template<typename Integer , typename UnsignedInteger = Integer> UnsignedInteger desalvo_standard_library::gcd (Integer *a*, Integer *b*)

```
Computes the greatest common divisor using Euclid's algorithm.
```

Parameters

a	is an input integer
b	is an input integer

Returns

the gcd(a,b)

```
#include "desalvo/numerical.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

std::cout << "dsl::gcd(5,3) = " << dsl::gcd(5,3) << std::endl;
std::cout << "dsl::gcd(-5,3) = " << dsl::gcd(-5,3) << std::endl;
std::cout << "dsl::gcd(5,-3) = " << dsl::gcd(5,-3) << std::endl;
std::cout << "dsl::gcd(-5,-3) = " << dsl::gcd(-5,-3) << std::endl;
std::cout << "dsl::gcd(0,3) = " << dsl::gcd(0,3) << std::endl;
std::cout << "dsl::gcd(-4,0) = " << dsl::gcd(-4,0) << std::endl;
std::cout << "dsl::gcd(2,12) = " << dsl::gcd(2,12) << std::endl;
std::cout << "dsl::gcd(1111111115,5) = " << dsl::gcd(1111111115,5) << std::endl;
std::cout << "dsl::gcd(54321,101) = " << dsl::gcd(54321,101) << std::endl;

return 0;
}
```

Should produce output

```
dsl::gcd(5,3) = 1
dsl::gcd(-5,3) = 1
dsl::gcd(5,-3) = 1
dsl::gcd(-5,-3) = 1
dsl::gcd(0,3) = 0
dsl::gcd(-4,0) = 0
dsl::gcd(2,12) = 2
dsl::gcd(1111111115,5) = 5
dsl::gcd(54321,101) = 1
```

6.1.3.17 template<typename T > T desalvo_standard_library::gcd (T *a*, T *b*)

Output operator which prints out top/bottom

Parameters

out	is the stream object
frac	is the Fraction object to output

Returns

a reference to the stream object for chaining Calculate the gcd of two unsigned integers

Parameters

a	is the left side
b	is the right side

Returns

gcd(a,b), i.e., the greatest common divisor.

6.1.3.18 bool desalvo_standard_library::getline (file< file_type::input > & *fin*, std::string & *s*)

```
Gets a line from the file
```

Parameters

fin	is the File object
s	stores the line from the file

Returns

true/false according to getline acting on the stream

```
// Program to generate rectangular version of Hilbert matrix
#include "std_cin.h"
#include "std_cout.h"
#include "file.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Local file path as std::string for easy concatenation
std::string prefix = "/Users/stephendesalvo/Documents/";

dsl::file<dsl::file_type::input> text(prefix + "
      data_richard_iii_opening_monologue.txt");

std::vector<std::string> v(5);

std::cout << "Let's get the first few lines of Richard III's opening monologue:" << std::endl;
dsl::getline(text, v[0]);
dsl::getline(text, v[1]);
dsl::getline(text, v[2]);
dsl::getline(text, v[3]);
dsl::getline(text, v[4]);

std::cout << v << std::endl;

return 0;
}
```

Should produce output

```
Let's get the first few lines of Richard III's opening monologue:
{Now is the winter of our discontent,Made glorious summer by this sun of York;,And all the clouds that lour
      'd upon our house,In the deep bosom of the ocean buried.,Now are our brows bound with victorious wreaths;}
```

6.1.3.19 template<typename String > bool desalvo_standard_library::getline (file< file_type::console > & *fin*, String & *s*)

```
Gets a line from the file
```

Parameters

fin	is the File object
s	stores the line from the file

Returns

true/false according to getline acting on the stream

```
// Program to generate rectangular version of Hilbert matrix
#include "std_cin.h"
```

```
#include "std_cout.h"
#include "file.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

dsl::file<dsl::file_type::console> console;

std::vector<std::string> v(5);

std::cout << "Let's try to recall the first few lines of Hamlet's to be or not to be speach" << std::endl;
dsl::getline(console, v[0]);
dsl::getline(console, v[1]);
dsl::getline(console, v[2]);
dsl::getline(console, v[3]);
dsl::getline(console, v[4]);

std::cout << "Let's see that again.\n";
std::cout << v << std::endl;

return 0;
}
```

Should produce output like the following:

```
Let's try to recall the first few lines of Hamlet's to be or not to be speach
To be, or not to be, that is the question
whether tis nobler in the mind to suffer
the slings and arrows of outrageous fortune
or to take arms against a sea of troubles
and by opposing, end them.  To die, to sleep
Let's see that again.
{To be, or not to be, that is the question,whether tis nobler in the mind to suffer,the slings and arrows
       of outrageous fortune,or to take arms against a sea of troubles,and by opposing, end them.  To die, to sleep}
```

6.1.3.20 template<typename Container , typename BinaryPredicate = std::equal_to<typename Container::value_type>, typename Comparison = std::less<typename Container::value_type>> bool desalvo_standard_library::has_unique_elements (Container *elements*, BinaryPredicate *pred* = std::equal_to<typename Container::value_type>(), Comparison *cmp* = std::less<typename Container::value_type>())

This sorts a copy of the container and then applies unique to test for uniqueness. Not the greatest algorithm.

Template Parameters

Container	is a container of elements
BinaryPredicate	is any function object which has operator(T,T)->bool overloaded, where T=Container::value_type is the type of object contained in objects of type Container
Comparison	is any function object which has operator< or operator(T,T)->bool that compares using less operation

Parameters

elements	is the collection of objects
pred	is a function object with operator(T,T)->bool overloaded

```
#include "desalvo/std_cout.h"
#include "desalvo/numerical.h" // See documentation for list of keywords included in
       this file.

namespace dsl = desalvo_standard_library;


int main(int argc, const char * argv[]) {

std::vector<int> v {1,2,3,4,5,6,7,8,9};
std::vector<int> v2 {1,9,2,8,3,7,4,6,5};
std::vector<int> v3 {4,3,5,2,6,7,3};

// WARNING!  There is no check against values outside of the values 1,...,n
```

```
std::vector<int> v4 {-1,4,-1};

std::cout << v << " has unique elements: " << (dsl::has_unique_elements(v) ? "yes"
       : "no") << std::endl;
std::cout << v2 << " has unique elements: " << (dsl::has_unique_elements(v2) ? "yes
       " : "no")<< std::endl;
std::cout << v3 << " has unique elements: " << (dsl::has_unique_elements(v3) ? "yes
       " : "no")<< std::endl;
std::cout << v4 << " has unique elements: " << (dsl::has_unique_elements(v4) ? "yes
       " : "no")<< std::endl;

// Generate all permutations of {-1,4,-1}
std::cout << std::endl;
auto sv = dsl::permutations(v4);
dsl::print(sv, "\n\n");

// Check if any pair of the permutations are the same coordinate-wise...should be 0.
std::cout << "Do any of those pairs of permutations have a difference whose sum of squares is 0? " << (
       dsl::has_unique_elements(sv, [](const std::vector<int>& a, const std::vector<int>&
       b) {
int ss = 0; // sum of squares initialize
for(size_t i=0,n=a.size();i<n;++i)
ss += (a[i]-b[i])*(a[i]-b[i]);
return ss!=0;
}) ? "yes" : "no") << std::endl << std::endl;

// Check if all pairs of the permutations have the same sum of squares...should be 1.
std::cout << "Do any of those permutations have the same sum of squares? " << (
       dsl::has_unique_elements(sv, [](const std::vector<int>& a, const std::vector<int>&
       b)->bool {
int ssa = 0; // sum of squares initialize
int ssb = 0;
for(size_t i=0,n=a.size();i<n;++i) {
ssa += a[i]*a[i];
ssb += b[i]*b[i];
}
return ssa!=ssb;
}) ? "yes" : "no") << std::endl;


return 0;
}
```

Should produce output

```
{1,2,3,4,5,6,7,8,9} has unique elements: yes
{1,9,2,8,3,7,4,6,5} has unique elements: yes
{4,3,5,2,6,7,3} has unique elements: no
{-1,4,-1} has unique elements: no

{{4,-1,-1},{-1,4,-1},{-1,-1,4},{-1,-1,4},{4,-1,-1},{-1,4,-1}}

Do any of those pairs of permutations have a difference whose sum of squares is 0? no

Do any of those permutations have the same sum of squares? yes
```

6.1.3.21 template<typename IntegerType , typename ContainerType = std::vector<IntegerType>> ContainerType desalvo_standard_library::int_to_digits (IntegerType *a*, bool *left_to_right* = `true`)

Converts the digits in the input into single characters in a collection of numbers

Template Parameters

IntegerType	is any unsigned integer type
ContainerType	is the container to store digits

Parameters

a	is the input value

Returns

a collection of digits of a stored individually

```
#include "desalvo/std_cout.h"
#include "desalvo/numerical.h" // See documentation for list of keywords included in
       this file.

namespace dsl = desalvo_standard_library;


int main(int argc, const char * argv[]) {

int x = 314159265;
auto v = dsl::int_to_digits(x);
std::cout << v << std::endl;

v = dsl::int_to_digits(x, false);
std::cout << v << std::endl;

// Just watch out for overflow!
int x2 = 314159265358979;
auto v2 = dsl::int_to_digits(x2);
dsl::print(v2);

return 0;
}
```

Should produce output (depending on numerical limits)

```
{3,1,4,1,5,9,2,6,5}
{5,6,2,9,5,1,4,1,3}
{4,1,2,4,7,4,2,3,7}
```

6.1.3.22 template<typename V > void desalvo_standard_library::iota (V & *v*, typename V::value_type *val* = `static_cast<typename V::value_type>(1)`)

iota

Template Parameters

V	is a container with a forward iterator, and property value_type

Parameters

v	is a container of values

val	is the initial value, which is statically cast to T

```
#include "desalvo/dsl_algorithm.h"
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

std::vector<char> small_letters() {
std::vector<char> v(26);
dsl::iota( v, 'a');
return v;
}

std::vector<char> capital_letters() {
std::vector<char> v(26);
dsl::iota( v, 'A');
return v;
}

int main(int argc, const char * argv[]) {

std::vector<int> v(10);
std::vector<char> v2(10);

dsl::iota(v,0);
dsl::print(v,"\n");

dsl::iota(v,1);
dsl::print(v,"\n");

// the second input is cast to V::value_type, so class of v must have this property defined
dsl::iota(v,-4.5);
dsl::print(v,"\n");

dsl::iota(v2, 'a');
dsl::print(v2, "\n");

dsl::print(small_letters(),"\n");
dsl::print(capital_letters(),"\n");


return 0;
}
```

Should produce output

```
{0,1,2,3,4,5,6,7,8,9}
{1,2,3,4,5,6,7,8,9,10}
{-4,-3,-2,-1,0,1,2,3,4,5}
{a,b,c,d,e,f,g,h,i,j}
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z}
{A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z}
```

6.1.3.23 template<typename Iterator , typename IntegerType > bool desalvo_standard_library::is_permutation_of_n (Iterator *start,* const Iterator & *stop,* IntegerType *n*)

Specifically checks if n numbers {a,b,c,d,...} are a permutation of {1,2,3,...,n}. There are very fast checksum ways of determining this, otherwise there is a generic algorithm that will work for larger n and even for more general objects

Template Parameters

Iterator	is the input iterator type
IntegerType	is the type of n

Parameters

start	refers to the first element
stop	refers to one after the last element
n	is the largest value in the set

```
#include "desalvo/std_cout.h"
```

```
#include "desalvo/numerical.h" // See documentation for list of keywords included in
       this file.

namespace dsl = desalvo_standard_library;


int main(int argc, const char * argv[]) {

std::vector<int> v {1,2,3,4,5,6,7,8,9};
std::vector<int> v2 {1,9,2,8,3,7,4,6,5};
std::vector<int> v3 {4,3,5,2,6,7,3};

// WARNING!  There is no check against values outside of the values 1,...,n
std::vector<int> v4 {-1,4,-1};

std::cout << v << " is permutation of 9: " << (dsl::is_permutation_of_n(std::begin(
      v), std::end(v), 9) ? "yes" : "no") << std::endl;
std::cout << v2 << " is permutation of 9: " << (dsl::is_permutation_of_n(std::begin
      (v2), std::end(v2), 9) ? "yes" : "no")<< std::endl;
std::cout << v3 << " is permutation of 7: " << (dsl::is_permutation_of_n(std::begin
      (v3), std::end(v3), 7) ? "yes" : "no")<< std::endl;
std::cout << v4 << " is permutation of 3: " << (dsl::is_permutation_of_n(std::begin
      (v4), std::end(v4), 3) ? "yes" : "no")<< std::endl;

return 0;
}
```

Should produce output

```
{1,2,3,4,5,6,7,8,9} is permutation of 9: yes
{1,9,2,8,3,7,4,6,5} is permutation of 9: yes
{4,3,5,2,6,7,3} is permutation of 7: no
{-1,4,-1} is permutation of 3: no
```

6.1.3.24 template<typename UnsignedIntegers > bool desalvo_standard_library::is_unique_uints_max_31 (UnsignedIntegers *values*)

Code taken from Cracking The Code Interview book. Very fast, cryptic.

Template Parameters

UnsignedIntegers	is any collection with forward iterator

Parameters

values	is the set of values to check for

Returns

true if all characters are unique from 0 to 31

```
#include "desalvo/std_cout.h"
#include "desalvo/numerical.h"

namespace dsl = desalvo_standard_library;


int main(int argc, const char * argv[]) {

std::vector<int> v {1,2,3,4,5,6,7,8,9};
std::vector<int> v2 {1,9,2,8,3,7,4,6,5};
std::vector<int> v3 {4,3,5,2,6,7,3};
std::vector<int> v4 {4,0,5,-2,6,7,3};

std::cout << dsl::is_unique_uints_max_31(v) << std::endl;
std::cout << dsl::is_unique_uints_max_31(v2) << std::endl;
std::cout << dsl::is_unique_uints_max_31(v3) << std::endl;

// one element is negative, so unpredictable behavior
std::cout << dsl::is_unique_uints_max_31(v4) << std::endl;

return 0;
}
```

Should produce output

```
1
1
0
1
```

6.1.3.25 template<typename ForwardIterator > bool desalvo_standard_library::is_unique_uints_max_31 (ForwardIterator *first*, ForwardIterator *last*)

Code taken from Cracking The Code Interview book. Very fast, cryptic.

Template Parameters

ForwardIterator	is any collection with input iterator

Parameters

values	is the set of values to check for

Returns

true if all characters are unique from 0 to 31

```
#include "desalvo/std_cout.h"
#include "desalvo/numerical.h"

namespace dsl = desalvo_standard_library;


int main(int argc, const char * argv[]) {

std::vector<int> v {1,2,3,4,5,6,7,8,9};
std::vector<int> v2 {1,9,2,8,3,7,4,6,5};
std::vector<int> v3 {4,3,5,2,6,7,3};
std::vector<int> v4 {4,0,5,-2,6,7,3};

std::cout << dsl::is_unique_uints_max_31(std::begin(v), std::end(v)) <<
      std::endl;
std::cout << dsl::is_unique_uints_max_31(std::begin(v2), std::end(v2)) <<
      std::endl;
std::cout << dsl::is_unique_uints_max_31(std::begin(v3), std::end(v3)) <<
      std::endl;

// one element is negative, so unpredictable behavior
std::cout << dsl::is_unique_uints_max_31(std::begin(v4), std::end(v4)) <<
      std::endl;

return 0;
}
```

Should produce output

```
1
1
0
1
```

6.1.3.26 std::vector< std::vector<short> > desalvo_standard_library::multiset_subsets (short *n*, short *k*)

Return all MULTISET subsets of [n] = {1,2,...,n} of size k, i.e., {{1,1,...,1},{1,1,...,1,2},...,{1,1,...,1,3},...{n,...,n}}

Parameters

n	is the set of values
k	is the subset size

Returns

all subsets of size k from [n]

```
#include "desalvo/numerical.h"
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

auto s0 = dsl::multiset_subsets(3,0);
auto s1 = dsl::multiset_subsets(3,1);
auto s2 = dsl::multiset_subsets(3,2);
auto s3 = dsl::multiset_subsets(3,3);

dsl::print(s0,"\n");
dsl::print(s1,"\n");
dsl::print(s2,"\n");
dsl::print(s3,"\n");

return 0;
}
```

Should produce output

```
{}
{{1},{2},{3}}
{{1,1},{1,2},{1,3},{2,1},{2,2},{2,3},{3,1},{3,2},{3,3}}
{{1,1,1},{1,1,2},{1,1,3},{1,2,1},{1,2,2},{1,2,3},{1,3,1},{1,3,2},{1,3,3},{2,1,1},{2,1,2},{2,1,3},{2,2,1},{2
      ,2,2},{2,2,3},{2,3,1},{2,3,2},{2,3,3},{3,1,1},{3,1,2},{3,1,3},{3,2,1},{3,2,2},{3,2,3},{3,3,1},{3,3,2},{3,3,3
      }}
```

6.1.3.27 template<typename V > bool desalvo_standard_library::next_permutation (V & *v*)

next_permutation

Template Parameters

V	is a container with a forward iterator, and property value_type

Parameters

v	is a container of values

Returns

whether values restarted

```
#include "desalvo/dsl_algorithm.h"
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

std::vector<char> small_letters(size_t n=26) {
std::vector<char> v(n);
dsl::iota( v, 'a');
return v;
}

std::vector<char> capital_letters(size_t n=26) {
std::vector<char> v(n);
dsl::iota( v, 'A');
return v;
}

int main(int argc, const char * argv[]) {

// generate first 5 letters a,b,c,d,e
auto v = small_letters(5);
dsl::print(v,"\n");

// Print all permutations of a,b,c,d,e
```

```
while( dsl::next_permutation(v))
dsl::print(v,"\n");

// generate first 5 CAPITAL letters A,B,C,D,E
auto v2 = capital_letters(5);

dsl::next_permutation(v2,std::greater<char>());
dsl::print(v2,"\n");

// Print all permutations of a,b,c,d,e
while( dsl::next_permutation(v2,std::greater<char>()))
dsl::print(v2,"\n");

return 0;
}
```

Should produce output

```
{a,b,c,d,e}
{a,b,c,e,d}
{a,b,d,c,e}
{a,b,d,e,c}
(... a bunch of other permutations, don't take this line literally! ...)
{e,d,b,a,c}
{e,d,b,c,a}
{e,d,c,a,b}
{e,d,c,b,a}
{E,D,C,B,A}
{E,D,C,A,B}
{E,D,B,C,A}
{E,D,B,A,C}
(... a bunch of other permutations, don't take this line literally! ...)
{A,B,D,E,C}
{A,B,D,C,E}
{A,B,C,E,D}
{A,B,C,D,E}
```

6.1.3.28 template<typename V , typename Compare > bool desalvo_standard_library::next_permutation (V & *v*, Compare && *cmp*)

next_permutation

Template Parameters

V	is a container with a forward iterator, and property value_type
Compare	is any type which provides a Binary Predicate testing for less than inequality

Parameters

v	is a container of values
cmp	is the comparison function object

Returns

whether values restarted

```
#include "desalvo/dsl_algorithm.h"
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

std::vector<char> small_letters(size_t n=26) {
std::vector<char> v(n);
dsl::iota( v, 'a');
return v;
}

std::vector<char> capital_letters(size_t n=26) {
std::vector<char> v(n);
dsl::iota( v, 'A');
return v;
}

int main(int argc, const char * argv[]) {
```

```
// generate first 5 letters a,b,c,d,e
auto v = small_letters(5);
dsl::print(v,"\n");

// Print all permutations of a,b,c,d,e
while( dsl::next_permutation(v))
dsl::print(v,"\n");

// generate first 5 CAPITAL letters A,B,C,D,E
auto v2 = capital_letters(5);

dsl::next_permutation(v2,std::greater<char>());
dsl::print(v2,"\n");

// Print all permutations of a,b,c,d,e
while( dsl::next_permutation(v2,std::greater<char>()))
dsl::print(v2,"\n");

return 0;
}
```

Should produce output

```
{a,b,c,d,e}
{a,b,c,e,d}
{a,b,d,c,e}
{a,b,d,e,c}
(... a bunch of other permutations, don't take this line literally! ...)
{e,d,b,a,c}
{e,d,b,c,a}
{e,d,c,a,b}
{e,d,c,b,a}
{E,D,C,B,A}
{E,D,C,A,B}
{E,D,B,C,A}
{E,D,B,A,C}
(... a bunch of other permutations, don't take this line literally! ...)
{A,B,D,E,C}
{A,B,D,C,E}
{A,B,C,E,D}
{A,B,C,D,E}
```

6.1.3.29 template<typename T1 , typename T2 , typename F = T1> F desalvo_standard_library::nfallingk (T1 *n*, T2 *k*)

calculates n!/k! = n(n-1)...(n-k+1), intermediate calculations are done in F

Template Parameters

T	is the input type
F	is the output type

Parameters

n	is the larger value
k	is the smaller value

Returns

n!/k!

```
#include "desalvo/numerical.h" // See documentation for list of keywords included in
       this file.
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

// typdef for large unsigned integer
typedef unsigned long long ull;

int main(int argc, const char * argv[]) {

// We must be careful!  By default we get a signed integer type, which is not defined in cases of overflow
auto x = dsl::nfallingk(50,25);
std::cout << x << std::endl; // -1040187392   <-- overflow!
```

```
// the function is templated so that you can specify the data type for which the calculations will be
       performed, in this case an unsigned long long is large enough.
auto x2 = dsl::nfallingk<ull>(50,25);
std::cout << x2 << std::endl;   // 15560789850943651840   <-- Ok!

// Make sure you know what you are doing!  This answer returned is actually 30! % ULLONG_MAX.  This
       behavior is in fact well-defined for unsigned types in C++, but you should always be careful when doing
       calculations involving extremely fast-growing numerical values.
auto x3 = dsl::nfallingk<ull>(1000,50);
std::cout << x3 << std::endl;   // 10657768518172278784  <-- (1000)_(50) % ULLONG_MAX

return 0;
}
```

Should produce output (depending on numeric limits):

```
-1040187392
15560789850943651840
10657768518172278784
```

6.1.3.30 template<typename T > bool desalvo_standard_library::operator!= (const Fraction< T > & *lhs*, const Fraction< T > & *rhs*)

Tests for inequality

Parameters

lhs	is the left object
rhs	is the right object

Returns

!(lhs==rhs)

6.1.3.31 bool desalvo_standard_library::operator!= (const binary_integer & *lhs*, const binary_integer & *rhs*)

Compares for inequality, a != b

Parameters

rhs	is the right hand side of a != b

Returns

false if a equals b, true otherwise

6.1.3.32 binary_integer desalvo_standard_library::operator& (binary_integer *lhs*, const binary_integer & *rhs*)

Computes a & b bitwise, returns new binary_integer

Parameters

rhs	is the right hand side of a & b

Returns

a & b

6.1.3.33 template<typename T > Fraction<T> desalvo_standard_library::operator* (Fraction< T > *f*, const Fraction< T > & *f2*)

Multiplies two fractions together

Parameters

f	is the left object
f2	is the right object

Returns

a new fraction object consisting of the product

6.1.3.34 binary_integer desalvo_standard_library::operator* (binary_integer *lhs*, const binary_integer & *rhs*)

Computes a $*$ b, returns new binary_integer

Parameters

rhs	is the right hand side of a $*$ b

Returns

a $*$ b

6.1.3.35 template<typename T > Fraction<T> desalvo_standard_library::operator+ (Fraction< T > *f*, const Fraction< T > & *f2*)

Adds two fractions together

Parameters

f	is the left object
f2	is the right object

Returns

a new fraction object consisting of the sum

6.1.3.36 binary_integer desalvo_standard_library::operator+ (binary_integer *lhs*, const binary_integer & *rhs*)

Computes a + b, returns new binary_integer

Parameters

rhs	is the right hand side of a + b

Returns

a + b

6.1.3.37 template<typename T > Fraction<T> desalvo_standard_library::operator- (Fraction< T > *f*, const Fraction< T > & *f2*)

Subtracts two fractions together

Parameters

f	is the left object
f2	is the right object

Returns

a new fraction object consisting of the difference

6.1.3.38 binary_integer desalvo_standard_library::operator- (binary_integer *lhs*, const binary_integer & *rhs*)

Computes a - b, returns new binary_integer

Parameters

rhs	is the right hand side of a - b

Returns

a - b

6.1.3.39 template<typename T > Fraction<T> desalvo_standard_library::operator/ (Fraction< T > *f*, const Fraction< T > & *f2*)

Divides two fractions together

Parameters

f	is the left object
f2	is the right object

Returns

a new fraction object consisting of the quotient

6.1.3.40 template<typename T > bool desalvo_standard_library::operator< (const Fraction< T > & *lhs*, const Fraction< T > & *rhs*)

Cross multiplies and compares the resulting integers. WARNING! This is not the best method, as there may be overflow. It would be more numerically stable to reduce the fractions to lowest terms and compare the numerators and denominators directly.

Parameters

lhs	is the left object
rhs	is the right object

Returns

true if the cross products correspond to strict inequality, false otherwise

6.1.3.41 binary_integer desalvo_standard_library::operator<< (binary_integer *lhs*, const binary_integer & *rhs*)

Bit shifts a up by b, returns new binary_integer

Parameters

rhs	is the right hand side of a << b

Returns

a shifted up by b

6.1.3.42 template<typename T > bool desalvo_standard_library::operator<= (const Fraction< T > & *lhs*, const Fraction< T > & *rhs*)

Returns

!(lhs>rhs)

6.1.3.43 bool desalvo_standard_library::operator<= (const binary_integer & *lhs*, const binary_integer & *rhs*)

Checks whether a <= b

Parameters

rhs	is the right hand side of a <= b

Returns

true if a <= b

6.1.3.44 template<typename T > bool desalvo_standard_library::operator== (const Fraction< T > & *lhs*, const Fraction< T > & *rhs*)

Cross multiplies and compares the resulting integers. WARNING! This is not the best method, as there may be overflow. It would be more numerically stable to reduce the fractions to lowest terms and compare the numerators and denominators directly.

Parameters

lhs	is the left object
rhs	is the right object

Returns

true if the cross products are the same, false otherwise

6.1.3.45 template<typename T > bool desalvo_standard_library::operator> (const Fraction< T > & *lhs*, const Fraction< T > & *rhs*)

Returns

!(rhs<lhs)

6.1.3.46 bool desalvo_standard_library::operator> (const binary_integer & *lhs*, const binary_integer & *rhs*)

Checks whether a > b

Parameters

rhs	is the right hand side of a > b

Returns

true if a > b

6.1.3.47 template<typename T > bool desalvo_standard_library::operator>= (const Fraction< T > & *lhs*, const Fraction< T > & *rhs*)

Returns

!(lhs<rhs)

6.1.3.48 bool desalvo_standard_library::operator>= (const binary_integer & *lhs*, const binary_integer & *rhs*)

Checks whether a >= b

Parameters

rhs	is the right hand side of a >= b

Returns

true if a >= b

6.1.3.49 binary_integer desalvo_standard_library::operator>> (binary_integer *lhs*, const binary_integer & *rhs*)

Bit shifts a down by b, returns new binary_integer

Parameters

rhs	is the right hand side of a >> b

Returns

a shifted down by b

6.1.3.50 binary_integer desalvo_standard_library::operator^ (binary_integer *lhs*, const binary_integer & *rhs*)

Computes a ^ b, returns new binary_integer

Parameters

rhs	is the right hand side of a ^ b

Returns

a ^ b

6.1.3.51 binary_integer desalvo_standard_library::operator| (binary_integer *lhs*, const binary_integer & *rhs*)

Computes a | b, returns new binary_integer

Parameters

rhs	is the right hand side of a | b

Returns

a | b

6.1.3.52 template<typename N = size_t, typename V = std::vector<N>, typename URNG = std::mt19937_64> V desalvo_standard_library::partial_permutation (N *n*, N *k*, URNG & *gen* = `generator_64`) `[inline]`

Returns a vector of size k of the first k random indices from the set {1,2,...,n}. O(n), or if k<sqrt(n) then $O({n} n)$ O(n)

Template Parameters

n	is the full size of the permutation
k	is the size of the partial permutation
gen	is the random generator, by default 64-bit

Returns

the first k values in a random permutation of n.

6.1.3.53 template<typename N = size_t, typename V = std::vector<N>, typename URNG = std::mt19937_64> V desalvo_standard_library::partial_permutation_rejection (N *n*, N *k*, URNG & *gen* = `generator_64`) `[inline]`

Returns a vector of size k of the first k random indices from the set {1,2,...,n} by selecting independent indices. NOT RECOMMENDED FOR k >> sqrt(n). Best for k = O(1). Duplicates are handled by the following algorithm:

1. Generate k independent indices
2. Sort indices using std::sort
3. remove duplicates using std::unique
4. regenerate indices until all are unique

O(k) O(k log(k))

Template Parameters

n	is the full size of the permutation
k	is the size of the partial permutation
gen	is the random generator, by default 64-bit

Returns

the first k values in a random permutation of n.

6.1.3.54 template<typename F = double, typename V = std::vector<F>> void desalvo_standard_library::partial_sum_in_place (V & *v*)

replaces the values with the partial sums, needs begin and end defined

Template Parameters

F	is the value type
V	is the container type

Parameters

v	is the container to sort elements in place.

```
#include "desalvo/numerical.h" // See documentation for list of keywords included in
      this file.
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

std::vector<int> v {5,3,4,2,5,6};
std::vector<std::valarray<int>> vs {{1,2,3,1,5,3},{4,4,4,3,2},{1,2,3,4,5},{5,4,3,2,1}};

dsl::partial_sum_in_place(v);
dsl::partial_sum_in_place(vs);

std::cout << v << std::endl;
std::cout << vs << std::endl;

return 0;
}
```

Should produce output:

```
{5,8,12,14,19,25}
{{1,2,3,1,5,3},{5,6,7,4,7,3},{6,8,10,8,12,3},{11,12,13,10,13,3}}
```

6.1.3.55 template<typename IntType = size_t, typename Container = std::vector<std::vector<IntType>>> Container desalvo_standard_library::permutation_as_product_of_cycles (const std::vector< IntType > & *permutation*)

Writes a permutation as a product of cycles

Template Parameters

IntType	is the underlying type of object
Container	is the container to hold the set of permutations

Parameters

permutation	is a permutation

Returns

a collection of values, each element in the collection is a cycle

6.1.3.56 template<typename IntType , typename Container = std::vector<std::vector<IntType>>> Container desalvo_standard_library::permutation_as_product_of_transpositions (const std::vector< IntType > & *permutation*)

Returns a list of transpositions of two elements, which when performed consecutively, starts with the identity permutation and gives the permutation input

Template Parameters

IntType	is the underlying type of object
Container	is the container to hold the set of permutations

Parameters

permutation	is a permutation

Returns

a collection of values, each element in the collection is a tranposition of two elements, i.e., tells which indices to swap

6.1.3.57 std::vector<size_t> desalvo_standard_library::permutation_reduction (std::vector< size_t > *vals*)

Reduces a permutation of the form {2, 40, 18, 12} –> {1,4,3,2}

Parameters

vals	is the vector of values

Returns

the vector of values in reduced form

Example:

6.1.3.58 template<typename T > std::vector< std::vector<T> > desalvo_standard_library::permutations (std::vector< T > *objects*)

Calculates and returns the set of all permutations as a vector of vectors.

Template Parameters

T	is the type of object

Parameters

objects	is some collection of objects

Returns

the set of all permutations of objects.

```
#include "desalvo/std_cout.h"
#include "desalvo/numerical.h" // See documentation for list of keywords included in
      this file.
```

```
namespace dsl = desalvo_standard_library;


int main(int argc, const char * argv[]) {

std::vector<int> v {3,1,4,5};

auto x = dsl::permutations(v);

dsl::print(x); // equivalent to std::cout << x;
dsl::print("\n\n\n");

dsl::sort_in_place(x);
dsl::print(x);


return 0;
}
```

Should produce output

```
{{1,4,5,3},{4,1,5,3},{4,5,1,3},{5,4,1,3},{1,5,4,3},{5,1,4,3},{5,4,3,1},{4,5,3,1},{4,3,5,1},{3,4,5,1},{5,3,4
      ,1},{3,5,4,1},{1,5,3,4},{5,1,3,4},{5,3,1,4},{3,5,1,4},{1,3,5,4},{3,1,5,4},{1,4,3,5},{4,1,3,5},{4,3,1,5},{3,4
      ,1,5},{1,3,4,5},{3,1,4,5}}


{{1,3,4,5},{1,3,5,4},{1,4,3,5},{1,4,5,3},{1,5,3,4},{1,5,4,3},{3,1,4,5},{3,1,5,4},{3,4,1,5},{3,4,5,1},{3,5,1
      ,4},{3,5,4,1},{4,1,3,5},{4,1,5,3},{4,3,1,5},{4,3,5,1},{4,5,1,3},{4,5,3,1},{5,1,3,4},{5,1,4,3},{5,3,1,4},{5,3
      ,4,1},{5,4,1,3},{5,4,3,1}}
```

6.1.3.59 template<typename V = std::vector<bool>, typename T = std::valarray<double>, typename N = size_t, typename URNG = std::mt19937_64, typename F = double> V desalvo_standard_library::poisson_fixedsum_poisson_process (const T & *p*, N *k*, URNG & *generator* = generator_64)

Randomly sample Bernoulli's with different parameters conditional on having sum k using Poisson Process.

Parameters

p	is a vector of probabilities
k	is the number of 1s.

Template Parameters

gen	is the random generator, by default 64-bit

Returns

an ordered vector of n 0s and 1s, exactly k of which are 1s.
an ordered vector of n 0s and 1s, exactly k of which are 1s.

6.1.3.60 template<typename V > bool desalvo_standard_library::prev_permutation (V & *v*)

prev_permutation

Template Parameters

V	is a container with a forward iterator, and property value_type

Parameters

v	is a container of values

Returns

whether values restarted

```
#include "desalvo/dsl_algorithm.h"
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

std::vector<char> small_letters(size_t n=26) {
std::vector<char> v(n);
dsl::iota( v, 'a');
return v;
}

std::vector<char> capital_letters(size_t n=26) {
std::vector<char> v(n);
dsl::iota( v, 'A');
return v;
}

int main(int argc, const char * argv[]) {

// generate first 5 letters a,b,c,d,e
auto v = small_letters(5);
dsl::prev_permutation(v);

// Print all permutations of a,b,c,d,e
while( dsl::prev_permutation(v))
dsl::print(v,"\n");

// generate first 5 CAPITAL letters A,B,C,D,E
auto v2 = capital_letters(5);
dsl::print(v2,"\n");

// Print all permutations of a,b,c,d,e
while( dsl::prev_permutation(v2,std::greater<char>()))
dsl::print(v2,"\n");

return 0;
}
```

Should produce output

```
{e,d,c,a,b}
{e,d,b,c,a}
{e,d,b,a,c}
{e,d,a,c,b}
(... a bunch of other permutations, don't take this line literally! ...)
{a,b,d,e,c}
{a,b,d,c,e}
{a,b,c,e,d}
{a,b,c,d,e}
{A,B,C,D,E}
{A,B,C,E,D}
{A,B,D,C,E}
{A,B,D,E,C}
(... a bunch of other permutations, don't take this line literally! ...)
{E,D,B,A,C}
{E,D,B,C,A}
{E,D,C,A,B}
{E,D,C,B,A}
```

6.1.3.61 template<typename V , typename Compare > bool desalvo_standard_library::prev_permutation (V & *v*, Compare *cmp*)

prev_permutation

Template Parameters

V	is a container with a forward iterator, and property value_type
Compare	is any type which provides a Binary Predicate testing for less than inequality

Parameters

v	is a container of values
cmp	is the comparison function object

Returns

whether values restarted

```
#include "desalvo/dsl_algorithm.h"
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

std::vector<char> small_letters(size_t n=26) {
std::vector<char> v(n);
dsl::iota( v, 'a');
return v;
}

std::vector<char> capital_letters(size_t n=26) {
std::vector<char> v(n);
dsl::iota( v, 'A');
return v;
}

int main(int argc, const char * argv[]) {

// generate first 5 letters a,b,c,d,e
auto v = small_letters(5);
dsl::prev_permutation(v);

// Print all permutations of a,b,c,d,e
while( dsl::prev_permutation(v))
dsl::print(v,"\n");

// generate first 5 CAPITAL letters A,B,C,D,E
auto v2 = capital_letters(5);
dsl::print(v2,"\n");

// Print all permutations of a,b,c,d,e
while( dsl::prev_permutation(v2,std::greater<char>()))
dsl::print(v2,"\n");

return 0;
}
```

Should produce output

```
{e,d,c,a,b}
{e,d,b,c,a}
{e,d,b,a,c}
{e,d,a,c,b}
(... a bunch of other permutations, don't take this line literally! ...)
{a,b,d,e,c}
{a,b,d,c,e}
{a,b,c,e,d}
{a,b,c,d,e}
{A,B,C,D,E}
{A,B,C,E,D}
{A,B,D,C,E}
{A,B,D,E,C}
(... a bunch of other permutations, don't take this line literally! ...)
{E,D,B,A,C}
{E,D,B,C,A}
{E,D,C,A,B}
{E,D,C,B,A}
```

6.1.3.62 template<typename T , typename String = std::string> void desalvo_standard_library::print (T && *container*, std::string *ending* = "", std::ostream & *out* = `std::cout`, String *separation* = `std::string(",")`, String *open_bracket* = `std::string("{")`, String *close_bracket* = `std::string("}")`)

outputs the elements in a container in a default, hopefully intuitive manner, without worrying about iterators or internal data structures.

Template Parameters

container	accepts all objects of any type (but not function pointers), it is of type universal reference

Parameters

ending	is a string to append to the output stream after the function finishes
out	is the output stream.

```
#include "desalvo/std_cout.h"
namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

std::multiset<int> v  {1,2,3,4,5};
std::list<int> v2  {0,0};
std::set<int> v3  {-1,2,-1234};
std::vector<int> v4 {3,1,4,1,5,9};
auto v5 {2,7,1,8,2,8};

dsl::print(v,"\n");
dsl::print(v2,"\n");
dsl::print(v3,"\n");
dsl::print(v4,"\n");
dsl::print(v5,"\n");

return 0;
}
```

Should produce output

```
{1,2,3,4,5}
{0,0}
{-1234,-1,2}
{3,1,4,1,5,9}
{2,7,1,8,2,8}
```

6.1.3.63 template<typename V , typename C > void desalvo_standard_library::print_side_by_side (const V & *left*, const C & *right*, const std::string & *sep* = `std::string("   ")`, const std::string & *endline* = `std::string("\n")`) `[inline]`

Simply prints the elements of two containers side-by-side separated by some string separator, where the second container needs at least as many elements as teh first container. Needs InputIterator.

Template Parameters

V	is a container with an input iterator
C	is a container with an input iterator

Parameters

left	is a container of elements for the left column
right	is a container of elements for the right column
sep	is the string that separates the two entries per line.

endline	is the string that separates each pair of entries.

```
#include "desalvo/numerical.h" // See documentation for list of keywords included in
       this file.
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;


int main(int argc, const char * argv[]) {

std::vector<int> v {3,1,4,1,5,9,2,6,5};
std::vector<double> v2 {3.1,4.1,5.9,2.6,5.3,5.8,9.7,9.3,2.3,8.6};
std::set<double> s;

// insert elements of v2 into a set s, which orders them
for(auto& x : v2) s.insert(x);

// v2 needs to have as many elements as v, prints as many elements as there are in v.
dsl::print_side_by_side(v,v2);
std::cout << std::endl << std::endl;

// v and s need only provide input iterators
dsl::print_side_by_side(v,s);


return 0;
}
```

Should produce output

```
3   3.1
1   4.1
4   5.9
1   2.6
5   5.3
9   5.8
2   9.7
6   9.3
5   2.3


3   2.3
1   2.6
4   3.1
1   4.1
5   5.3
9   5.8
2   5.9
6   8.6
5   9.3
```

6.1.3.64 template<typename InputIterator1 , typename InputIterator2 > void desalvo_standard_library::print_side_by_side (InputIterator1 *start1*, InputIterator1 *stop*, InputIterator2 *start2*, const std::string & *sep* = `std::string("   ")`, const std::string & *endline* = `std::string("\n")`)

Simply prints the elements of two containers side-by-side separated by some string separator, where the second container needs at least as many elements as teh first container. Needs InputIterator.

Template Parameters

InputIterator1	is any input iterator
InputIterator2	is any input iterator

Parameters

left	is a container of elements for the left column
right	is a container of elements for the right column
sep	is the string that separates the two entries per line.
endline	is the string that separates each pair of entries.

```
#include "desalvo/numerical.h" // See documentation for list of keywords included in
       this file.
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;


int main(int argc, const char * argv[]) {

std::vector<int> v {3,1,4,1,5,9,2,6,5};
std::vector<double> v2 {3.1,4.1,5.9,2.6,5.3,5.8,9.7,9.3,2.3,8.6};
std::set<double> s;

// insert elements of v2 into a set s, which orders them
for(auto& x : v2) s.insert(x);

// Can also input by iterators
dsl::print_side_by_side(std::begin(v)+3, std::end(v), std::begin(v2));
std::cout << std::endl << std::endl;

// Can mix and match iterators of different types, as long as input iterators
dsl::print_side_by_side(std::begin(v), std::end(v)-5, std::begin(s));


return 0;
}
```

Should produde output

```
1   3.1
5   4.1
9   5.9
2   2.6
6   5.3
5   5.8


3   2.3
1   2.6
4   3.1
1   4.1
```

6.1.3.65 template<typename T = bool, typename Vector = std::vector<T>, typename URNG = std::mt19937_64> Vector desalvo_standard_library::random_binary_row (size_t *n*, size_t *k*, T *val* = `true`, URNG & *gen* = `generator_64`)

Returns a random permutation of the vector (val,val,...,val,[0],[0],...,[0]) of k vals and n-k [0]s, where [0] is the default value of the container.

Parameters

n	is the size of the vector
k	is the number of vals
val	is the value to fill in
gen	is the uniform random number generator

Template Parameters

T	is the type of each element
Vector	is the type which stores the collection of T objects
URNG	is the type of the random number generator

Returns

(val,val,...,val,[0],[0],...,[0]) of k vals and n-k [0]s

6.1.3.66 template<typename T > T desalvo_standard_library::random_integer (T *a*, T *b*)

Quick random integer using 64-bit default generator

Template Parameters

a	is the lower bound
b	is the upper bound

Returns

random number in {a,a+1,...,b-1,b}

6.1.3.67 template<typename T , typename V = std::vector<T>> V desalvo_standard_library::random_integer_vector (T *a*, T *b*, size_t *n*)

Quick random integer using 64-bit default generator

Template Parameters

a	is the lower bound
b	is the upper bound

Returns

random number in {a,a+1,...,b-1,b}

6.1.3.68 template<typename N = std::size_t, typename V = std::vector<N>, typename URNG = std::mt19937_64> V desalvo_standard_library::random_permutation (N *n*, URNG & *gen* = `generator_64`)

Quickly generate a random permutation of numbers {1,...,n} using default 64–bit generator

Template Parameters

n	is the largest number
gen	is the random number generator, default at 64-bits

Returns

a permutation of elements 1 through n

6.1.3.69 template<typename N = std::size_t, typename V = std::vector<N>, typename URNG = std::mt19937_64> V desalvo_standard_library::random_permutation_fixed_point_free (N *n*, URNG & *gen* = `generator_64`)

Quickly generate a random permutation of numbers {1,...,n} using default 64–bit generator

Template Parameters

N	is the data type of values, typically integer type
V	is the container for the collection of values of type N
URNG	is the random number generator type, by default mt19937_64

Parameters

n	is the largest number
gen	is the random number generator, default at 64-bits

Returns

a permutation of elements 1 through n

6.1.3.70 template<typename N = size_t, typename Float = long double, typename V = std::vector<N>, typename URNG = std::mt19937_64> V desalvo_standard_library::random_permutation_mallows (N *n*, Float *q*, URNG & *gen* = `generator_64`)

Quickly generate a random permutation of numbers {1,...,n} using default 64–bit generator

Template Parameters

n	is the largest number
gen	is the random number generator, default at 64-bits

Returns

a permutation of elements 1 through n

6.1.3.71 template<typename N = size_t, typename Float = long double, typename V = std::vector<N>, typename URNG = std::mt19937_64> V desalvo_standard_library::random_permutation_mallows_in_mallows_form (N *n*, Float *q*, URNG & *gen* = `generator_64`)

```
Quickly generate a random permutation of numbers {1,...,n} using default 64--bit generator
```

Template Parameters

n	is the largest number
gen	is the random number generator, default at 64-bits

Returns

a permutation of elements 1 through n

```
Example 1:
```

```
#include "desalvo/dsl_usings.h" // See documentation for list of keywords included in
      this file.
#include <random>
#include <chrono>

    // avoiding 132
    template<typename T>
    bool contains_132_weakly(std::vector<T>& v) {

        size_t n = v.size();

        // Look at all triplets of indices to see if pattern is violated.
        // O(n^3) algorithm.
        for(size_t i=0;i<n-2;++i)
            for(size_t j=i+1;j<n-1;++j)
                for(size_t k=j+1;k<n;++k)
                    if( (v[i] < v[j] && v[i] < v[k] && v[j] >= v[k]) ){
                        std::cout << std::vector<size_t>({i,j,k}) << std::endl;
                        return true;
                    }

        return false;
    }


    int main(int argc, const char * argv[]) {

        auto v = dsl::random_permutation_mallows_in_mallows_form
      (10,1.);
        std::cout << v << std::endl;
        std::cout << contains_132_weakly(v) << std::endl;
```

```
        return 0;
    }
```

Example 2:

#include "desalvo/dsl_usings.h" // See documentation for list of keywords included in this file.

avoiding 132 template<typename T> bool contains_321_consecutively_weakly(const std::vector<T>& v) {

```
size_t n = v.size();
```

Look at all triplets of indices to see if pattern is violated. O(n) algorithm. for(size_t i=0;i<n-3;++i) if((v[i] >= v[i+1] && v[i+1] >= v[i+2])){ std::cout << std::vector<size_t>({i,j,k}) << std::endl; return true; } return true;

```
    return false;

}

int main(int argc, const char * argv[]) {
```

Create mesh grid double qmin = .01; double qmax = 4.; size_t mesh_size = 20; std::vector<double> vals(mesh_-size);

n is the size of the permutation, m is the number of iterations. size_t n = 20; size_t m = 100000;

keeps track of which value of q we are using in the vector size_t index = 0;

q = qmin, qmin+delta, qmin+2delta, ..., qmax for(long double q = qmin; q <= qmax; q += (qmax-qmin)/(mesh_size-1)) {

```
double avoids_321 = 0.; // count number that avoid 321

for(size_t i=0;i<m;++i)
```

generate permutation using Mallows(q) distribution, test for whether it avoids 321 consecutively. if(contains_321_-consecutively_weakly(dsl::random_permutation_mallows_in_mallows_form(n, q))) avoids_321 = avoids_321 + 1.;

Keep track of which ones avoid, store the average^(1/n) if(avoids_321) vals[index++] = std::pow(avoids_321/m,1./n); else vals[index++] = 1.; }

dsl::print(vals,"[", ",", "]");

return 0; }

6.1.3.72 template<typename N = size_t, typename Float = long double, typename URNG = std::mt19937_64> std::vector<N> desalvo_standard_library::random_permutation_mallows_ordering_construction (N *n*, Float *q*, URNG & *gen* = generator_64)

Returns a random permutation size n according to the Mallows(q) distribution.

Parameters

n	is the size of the permutation
q	is the positive, real-valued weight
gen	is the uniform random number generator

Template Parameters

N	is the type of the permutation elements
Float	is the type for q, which can be very sensitive for q small and q large
URNG	is the type of the random number generator

Returns

random permutation generated according to the Mallows(q) distribution

Examples:

```
#include <algorithm>
#include <numeric>

#include "std_cout.h"
#include "permutation.h"
#include "numerical.h"
#include "statistics.h"

namespace dsl = desalvo_standard_library;


size_t number_of_123(const std::vector<size_t>& v) {

size_t k = 0;

for(size_t i=0,n=v.size()-2;i<n;++i)
if(v[i]<v[i+1] && v[i+1]<v[i+2])
++k;

return k;
}



size_t number_of_2341(const std::vector<size_t>& v) {

size_t k = 0;

for(size_t i=0,n=v.size()-3;i<n;++i)
if(v[i+3]<v[i] && v[i+1]<v[i+2]&& v[i+2]<v[i+3])
++k;

return k;
}



size_t number_of_1243(const std::vector<size_t>& v) {

size_t k = 0;

for(size_t i=0,n=v.size()-3;i<n;++i)
if(v[i]<v[i+1]&& v[i+1]<v[i+3] && v[i+3] < v[i+2])
++k;

return k;
}

size_t number_of_1432(const std::vector<size_t>& v) {

size_t k = 0;

for(size_t i=0,n=v.size()-3;i<n;++i)
if(v[i]<v[i+3]&& v[i+3]<v[i+2] && v[i+2] < v[i+1])
++k;

return k;
}


size_t number_of_1342(const std::vector<size_t>& v) {

size_t k = 0;

for(size_t i=0,n=v.size()-3;i<n;++i)
if(v[i]<v[i+3]&& v[i+3]<v[i+1] && v[i+1] < v[i+2])
++k;

return k;
}


int main(int argc, const char * argv[]) {

//dsl::integer_partition_generator<unsigned int> ip_gen(100);

//auto t = ip_gen.recursive_method_table(5,5);
```

```
//std::cout << t << std::endl;


size_t imax = 10000;
size_t n = 1000;

auto fun = number_of_1342;

std::vector<size_t> counts(imax);

std::cout << "datapoint2 = ";
for(size_t i=0;i<imax;i++) {
std::vector<size_t> v = dsl::random_permutation_mallows_ordering_construction<size_t,long double>(n,.2);
counts[i]=fun(v);
}

std::cout << counts << std::endl;




counts.resize(imax,0);
std::cout << "datapoint5 = ";
for(size_t i=0;i<imax;i++) {
std::vector<size_t> v = dsl::random_permutation_mallows_ordering_construction<size_t,long double>(n,.5);
counts[i]=fun(v);
}

std::cout << counts << std::endl;

counts.resize(imax,0);
std::cout << "data1 = ";
for(size_t i=0;i<imax;i++) {
std::vector<size_t> v = dsl::random_permutation_mallows_ordering_construction<size_t,long double>(n,1.);
counts[i]=fun(v);
}

std::cout << counts << std::endl;



counts.resize(imax,0);
std::cout << "data1point3 = ";
for(size_t i=0;i<imax;i++) {
std::vector<size_t> v = dsl::random_permutation_mallows_ordering_construction<size_t,long double>(n,1.3);
counts[i]=fun(v);
}

std::cout << counts << std::endl;



counts.resize(imax,0);
std::cout << "data2 = ";
for(size_t i=0;i<imax;i++) {
std::vector<size_t> v = dsl::random_permutation_mallows_ordering_construction<size_t,long double>(n,2.);
counts[i]=fun(v);
}

std::cout << counts << std::endl;



counts.resize(imax,0);
std::cout << "data5 = ";
for(size_t i=0;i<imax;i++) {
std::vector<size_t> v = dsl::random_permutation_mallows_ordering_construction<size_t,long double>(n,5.);
counts[i]=fun(v);
}

std::cout << counts << std::endl;

return 0;
}
```

6.1.3.73 template<typename N = std::size_t, typename V = std::vector<N>, typename URNG = std::mt19937_64> V desalvo_standard_library::random_permutation_shifted (N *n*, N *a*, URNG & *gen* = `generator_64`)

Quickly generate a random permutation of numbers {a,a+1,...,a+n-1} using default 64–bit generator

Template Parameters

n	is the length of consecutive numbers
gen	is the random number generator, default at 64-bits

Returns

a permutation of elements {a,a+1,...,a_n-1}

6.1.3.74 template<typename F = double, typename V = std::vector<F>, typename Size = size_t> V desalvo_standard_library::range (Size *n*, F *initial_value*)

Initializes list to {initial_value, initial_value+1,...,initial_value+n-1}. By default, the initial value of 1.

Template Parameters

Size	is any nonnegative integer type
V	is the container to store the values
F	is the data type of the values

Parameters

n	is the size of the collection
initial_value	is the value of the first element.

Returns

a collection of values starting with initial_value and adding 1 n-1 times

```
// Example of using dsl::range
#include "desalvo/numerical.h" // See documentation for list of keywords included in
       this file.
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

    // range calls the += operator, so can be used with any numerical container as well like valarray
    auto v = dsl::range(10);
    auto v2 = dsl::range(10, 0.1);
    auto v3 = dsl::range(10, 0);
    auto v4 = dsl::range(10, std::valarray<short>({0,1,2,3,4}));

    std::cout << v << std::endl;
    std::cout << v2 << std::endl;
    std::cout << v3 << std::endl;
    std::cout << v4 << std::endl;

    return 0;
}
```

6.1.3.75 template<typename T , typename String = std::string> void desalvo_standard_library::read (T & *container*, std::istream & *in* = `std::cin`)

outputs the elements in a container in a default, hopefully intuitive manner, without worrying about iterators or internal data structures.

Template Parameters

container	accepts all objects of any type (but not function pointers), it is of type universal reference

Parameters

ending	is a string to append to the output stream after the function finishes
out	is the output stream.

```
#include "desalvo/std_cout.h"
namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

std::multiset<int> v  {1,2,3,4,5};
std::list<int> v2  {0,0};
std::set<int> v3  {-1,2,-1234};
std::vector<int> v4 {3,1,4,1,5,9};
auto v5 {2,7,1,8,2,8};

dsl::print(v,"\n");
dsl::print(v2,"\n");
dsl::print(v3,"\n");
dsl::print(v4,"\n");
dsl::print(v5,"\n");

return 0;
}
```

Should produce output

```
{1,2,3,4,5}
{0,0}
{-1234,-1,2}
{3,1,4,1,5,9}
{2,7,1,8,2,8}
```

6.1.3.76 template<typename V > void desalvo_standard_library::reverse_in_place (V & *v*)

Reverses elements

Template Parameters

V	is any bidirectional iterator

Parameters

v	is the container of elements

```
#include "desalvo/numerical.h" // See documentation for list of keywords included in
       this file.
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

std::vector<int> v {5,3,4,2,5,6};
std::vector<std::valarray<int>> vs {{1,2,3,1,5,3},{4,4,4,3,2},{1,2,3,4,5},{5,4,3,2,1}};

dsl::reverse_in_place(v);
dsl::reverse_in_place(vs);

std::cout << v << std::endl;
std::cout << vs << std::endl;

return 0;
}
```

Should produce output:

```
{6,5,2,4,3,5}
{{5,4,3,2,1},{1,2,3,4,5},{4,4,4,3,2},{1,2,3,1,5,3}}
```

6.1.3.77 template<typename T = bool, typename Vector = std::vector<T>, typename N = size_t, typename URNG = std::mt19937_64> Vector desalvo_standard_library::set_2n_choose_n (N *n*, URNG & *gen* = generator_64)

Randomly sample iid Bernoulli's conditional on having sum k. O(n) O(n log n) due to the shuffle operation

Parameters

n	is the number of Bernoulli random variables
k	is the number of 1s.

Template Parameters

gen	is the random generator, by default 64-bit

Returns

an ordered vector of n 0s and 1s, exactly k of which are 1s.

6.1.3.78 template<typename T = bool, typename Vector = std::vector<T>, typename N = size_t, typename URNG = std::mt19937_64> Vector desalvo_standard_library::set_n_choose_k (N *n*, N *k*, URNG & *gen* = generator_64)

```
Randomly sample iid Bernoulli's conditional on having sum k.
```

O(n) O(n) due to self-similar PDC

Parameters

n	is the number of Bernoulli random variables
k	is the number of 1s.

Template Parameters

gen	is the random generator, by default 64-bit

Returns

an ordered vector of n 0s and 1s, exactly k of which are 1s.

Example 1:

```
// Code to check the 6 choose 3 different sets, make sure each are occurring equally likely
#include "desalvo/dsl_usings.h" // See documentation for list of keywords included in
      this file.

int main(int argc, const char * argv[]) {

std::multiset< std::vector<bool> > s;

size_t m = 100000;

for(size_t i=0;i<m;++i) {


s.insert(dsl::set_n_choose_k(6,3));
}

std::cout << s.count( {1,1,1,0,0,0} ) << std::endl;
std::cout << s.count( {1,1,0,1,0,0} ) << std::endl;
std::cout << s.count( {1,1,0,0,1,0} ) << std::endl;
std::cout << s.count( {1,1,0,0,0,1} ) << std::endl;
std::cout << s.count( {1,0,1,1,0,0} ) << std::endl;
std::cout << s.count( {1,0,1,0,1,0} ) << std::endl;
std::cout << s.count( {1,0,1,0,0,1} ) << std::endl;
std::cout << s.count( {1,0,0,1,1,0} ) << std::endl;
```

```
 std::cout << s.count( {1,0,0,1,0,1} ) << std::endl;
 std::cout << s.count( {1,0,0,0,1,1} ) << std::endl;
 std::cout << s.count( {0,1,1,1,0,0} ) << std::endl;
 std::cout << s.count( {0,1,1,0,1,0} ) << std::endl;
 std::cout << s.count( {0,1,1,0,0,1} ) << std::endl;
 std::cout << s.count( {0,1,0,1,1,0} ) << std::endl;
 std::cout << s.count( {0,1,0,1,0,1} ) << std::endl;
 std::cout << s.count( {0,1,0,0,1,1} ) << std::endl;
 std::cout << s.count( {0,0,1,1,1,0} ) << std::endl;
 std::cout << s.count( {0,0,1,1,0,1} ) << std::endl;
 std::cout << s.count( {0,0,1,0,1,1} ) << std::endl;
 std::cout << s.count( {0,0,0,1,1,1} ) << std::endl;


return 0;
 }
```

Example 2:

```
#include "desalvo/dsl_usings.h" // See documentation for list of keywords included in
       this file.

int main(int argc, const char * argv[]) {

auto v = dsl::set_n_choose_k(1100,5);
std::cout << v << std::endl;

std::cout << std::count(std::begin(v), std::end(v), true) << std::endl;

return 0;
}
```

Example 3:

```
// Code to check the n choose k different sets, make sure each are occurring equally likely
#include "desalvo/dsl_usings.h" // See documentation for list of keywords included in
       this file.

int main(int argc, const char * argv[]) {

// from (0,0) to (n,k)
size_t n = 6;
size_t k = 3;
size_t m = 100000;

std::multiset< std::vector<bool> > s;

// Insert each generated set into s
for(size_t i=0;i<m;++i)
s.insert(dsl::set_n_choose_k(n,k));

// Generate all possible paths from (0,0) to (n,k)
dsl::north_east_lattice_path<bool> paths(n,k);

// Print out each path along with the number of times it occurs in s
for(auto& x : paths)
std::cout << x << ": " << s.count(x) << std::endl;

return 0;
}
```

6.1.3.79 template<typename V = std::vector<size_t>> V desalvo_standard_library::sieve (size_t *n*)

Creates a list of prime numbers up to n+1.

Template Parameters

V	is the container to store the elements

Parameters

n	is the upper bound on prime numbers

Returns

a list of prime numbers up to n+1.

```
#include "desalvo/numerical.h"
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

auto v = dsl::sieve(100);

// Make a list of all prime numbers up to 101
dsl::print(v);


return 0;
}
```

Should produce output

```
{2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101}
```

6.1.3.80 template<typename T , typename RandomAccess > void desalvo_standard_library::sort_between (RandomAccess *start*, RandomAccess *stop*, T *val*)

sorts elements between a given value, e.g., sorts elements between all instances of 2.

Parameters

start	is the initial location
stop	is one after last location
val	is the segmenting value

6.1.3.81 template<typename V , typename Comparison = std::less<typename V::value_type>> void desalvo_standard_library::sort_in_place (V & *v*, Comparison *cmp* = `std::less<typename V::value_type>()`)

In place sort, needs begin and end defined with random access iterator

Template Parameters

V	is the container type

Parameters

v	is the container to sort elements in place.

```
#include "desalvo/numerical.h" // See documentation for list of keywords included in
      this file.
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

std::vector<int> v {5,3,4,2,5,6};
std::vector<std::vector<int>> vs {{1,2,3,1,5,3},{4,4,4,3,2},{1,2,3,4,5},{5,4,3,2,1}};

dsl::sort_in_place(v);
dsl::sort_in_place(vs);

std::cout << v << std::endl;
std::cout << vs << std::endl;

return 0;
```

```
}
```

Should produce output:

```
{2,3,4,5,5,6}
{{1,2,3,1,5,3},{1,2,3,4,5},{4,4,4,3,2},{5,4,3,2,1}}
```

Another example

```
#include "desalvo/std_cout.h"
#include "desalvo/numerical.h" // See documentation for list of keywords included in
       this file.

namespace dsl = desalvo_standard_library;


int main(int argc, const char * argv[]) {

std::vector<int> v {3,1,4,5};

// Print in current order
dsl::print(v,"\n");

// decreasing order
dsl::sort_in_place(v, [](int a, int b) { return a>b; });

dsl::print(v,"\n");

// increasing order
dsl::sort_in_place(v);
dsl::print(v,"\n");


return 0;
}
```

Should produce output

```
{3,1,4,5}
{5,4,3,1}
{1,3,4,5}
```

6.1.3.82 template<typename ReturnValueType = double, typename IntegerType = long long int, typename DataType = ReturnValueType, typename InputIterator = typename std::vector<DataType>::iterator> ReturnValueType desalvo_standard_library::sum_of_powers (InputIterator *start,* InputIterator *stop,* IntegerType *power,* DataType *initial* = 0.)

Calculate the sums of powers of a collection of objects

Parameters

start	is an iterator to starting value
stop	is an iterator to one after the last value
initial	is the initial value to add to the sum

Template Parameters

power	is the exponent to raise the values to

Returns

x_1^a+x_2^a+...+x_n^a, the sums of powers of elements

```
#include "desalvo/numerical.h" // See documentation for list of keywords included in
       this file.
#include "desalvo/std_cout.h"
```

```
namespace dsl = desalvo_standard_library;


int main(int argc, const char * argv[]) {

std::vector<int> v {3,1,4,1,5,9,2,6,5};
std::vector<double> v2 {3.1,4.1,5.9,2.6,5.3,5.8,9.7,9.3,2.3,8.6};

// Sum of squares
auto x = dsl::sum_of_powers(std::begin(v), std::end(v), 2);

// Sum of 4th powers
auto x2 = dsl::sum_of_powers(std::begin(v2), std::end(v2), 4);

std::cout << x << std::endl;
std::cout << x2 << std::endl;

return 0;
}
```

Shoudl produce output

```
198
25384.6
```

6.1.3.83 template<typename Size = size_t, typename V = std::vector<std::pair<Size,Size>>> V desalvo_standard_library::table_indices (Size *m*, Size *n*, Size *initial_value_first* = 0, Size *initial_value_second* = 0)

```
Initializes list to {{i,j},{i+1,j},...,{i+m-1,j},{i,j+1},{i+1,j+1},...,{i+m-1,j+1},...,{i+m-1,j+n-1}}, where i
```

Can change list to {initial_value+1,...,initial_value+n-1} by second parameter initial_value

Template Parameters

Size	is any nonnegative integer type
V	is the container to store the values

Parameters

m	is the size of the collection
n	is the size of the collection
initial_value_first	is the first coordinate of the first index.
initial_value_-second	is the second coordinate of the first index.

Returns

an m x n collection of indices forming a rectangular region of a matrix

```
#include "desalvo/numerical.h" // See documentation for list of keywords included in
      this file.
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Form matrix [[1,2,3],[4,5,6],[7,8,9]] using a row-major 1D vector
std::vector<int> v {1,2,3,4,5,6,7,8,9};

auto two_by_two = dsl::table_indices(2,2);

auto start = std::begin(two_by_two);

// To access an element at index (i,j) in the array, use j+i*m, where m is the number of rows
std::cout << "Two by two submatrix of elements (1:2,1:2) \n";
std::cout << "[[" << v[start->second + start->first*3] << ",";          ++start;
std::cout          << v[start->second + start->first*3] << "],\n[";     ++start;
std::cout          << v[start->second + start->first*3] << ",";         ++start;
std::cout          << v[start->second + start->first*3] << "]]";        ++start;
```

```
return 0;
}
```

6.1.3.84 template<typename _InputIterator , typename Size , typename _OutputIterator , typename _UnaryOperation > void desalvo_standard_library::transform_n (_InputIterator __first, Size __n, _OutputIterator __result, _UnaryOperation __op)

Generic algorithm for apply, applies function to n elements

Template Parameters

_InputIterator	is the input iterator type
Size	is any unsigned integer type large enough to index values from 0,...,__n-1
_OutputIterator	is the output iterator type for result
_UnaryOperation	is the function that is applied to each element

Parameters

__first	is an iterator to the first of a collection of at least n elements
__n	is the number of elements for which to apply __op
__result	is an iterator to the first of a collection of at least n elements
__op	is the function object with unary function operator

```
#include "desalvo/std_cout.h"
#include "desalvo/numerical.h"

namespace dsl = desalvo_standard_library;


int main(int argc, const char * argv[]) {

std::vector<int> v {1,2,3,4,5,6,7,8,9,10};

// Square all numbers
dsl::print(v,"\n");
dsl::transform_n(std::begin(v), 10, std::begin(v), [](int& a) { return a*a; } );

// double the first five
dsl::print(v,"\n");
dsl::transform_n(std::begin(v), 5, std::begin(v), [](int& a) { return a+a; } );
dsl::print(v,"\n");

// Square all numbers, store the result in s
std::vector<int> s(10);
dsl::transform_n(std::begin(v), 10, std::begin(s), [](int& a) { return a*a; } );
dsl::print(s,"\n");


return 0;
}
```

Should produce output

```
{1,2,3,4,5,6,7,8,9,10}
{1,4,9,16,25,36,49,64,81,100}
{2,8,18,32,50,36,49,64,81,100}
{4,64,324,1024,2500,1296,2401,4096,6561,10000}
```

6.1.3.85 template<typename _InputIterator1 , typename Size , typename _InputIterator2 , typename _OutputIterator , typename _BinaryOperation > void desalvo_standard_library::transform_n (_InputIterator1 __first1, Size __n, _InputIterator2 __first2, _OutputIterator __result, _BinaryOperation __binary_op)

Generic algorithm for apply, applies function to n elements

Template Parameters

_InputIterator1	is the input iterator type of the first collection
_InputIterator2	is the input iterator type of the second collection
Size	is any unsigned integer type large enough to index values from 0,...,__n-1
_OutputIterator	is the output iterator type for result
_BinaryOperation	is the function that is applied to each element

Parameters

__first1	is an iterator to the first of a collection of at least n elements
__n	is the number of elements for which to apply __op
__first2	is an iterator to the first of a collection of at least n elements
__result	is an iterator to the first of a collection of at least n elements
__binary_op	is the function object with binary function operator

```
#include "desalvo/std_cout.h"
#include "desalvo/numerical.h"

namespace dsl = desalvo_standard_library;


int main(int argc, const char * argv[]) {

std::vector<int> v {1,2,3,4,5,6,7,8,9,10};

// Square all numbers
dsl::print(v,"\n");
dsl::transform_n(std::begin(v), 10, std::begin(v), [](int& a) { return a*a; } );

// double the first five
dsl::print(v,"\n");
dsl::transform_n(std::begin(v), 5, std::begin(v), [](int& a) { return a+a; } );
dsl::print(v,"\n");

// Square all numbers, store the result in s
std::vector<int> s(10);
dsl::transform_n(std::begin(v), 10, std::begin(s), [](int& a) { return a*a; } );
dsl::print(s,"\n");

// Go a little crazy
std::vector<int> s2(10);
dsl::transform_n(std::begin(v), 10, std::begin(s), std::begin(s2), [](int a, int b) {
      return a*b+b*b-4*a; } );
dsl::print(s2,"\n");



return 0;
}
```

Should produce output

```
{1,2,3,4,5,6,7,8,9,10}
{1,4,9,16,25,36,49,64,81,100}
{2,8,18,32,50,36,49,64,81,100}
{4,64,324,1024,2500,1296,2401,4096,6561,10000}
{16,4576,110736,1081216,6374800,1726128,5882254,17039104,43577838,100999600}
```

6.1.3.86 template<class RandomAccessIterator > void desalvo_standard_library::transpose (RandomAccessIterator *first*, RandomAccessIterator *last*, size_t *m*)

In place transposition of a row-major matrix of size m*n, assumes contiguous array. Code from http-://stackoverflow.com/questions/9227747/in-place-transposition-of-a-matrix

Written by Christian Ammer.

I changed int m third parameter to size_t m

Template Parameters

RandomAccessIterator	is any random access iterator

Parameters

first	is the iterator to first element
last	is the iterator to one after last element
m	is the number of columns

```
#include "desalvo/numerical.h" // See documentation for list of keywords included in
       this file.
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Form matrix [[1,2,3],[4,5,6],[7,8,9]] using a row-major 1D vector
std::vector<int> v {1,2,3,4,5,6,7,8,9};

dsl::print(v,"\n");

auto two_by_two = dsl::table_indices(2,2);
auto start = std::begin(two_by_two);

// To access an element at index (i,j) in the array, use j+i*m, where m is the number of rows
std::cout << "Two by two submatrix of elements (1:2,1:2) \n";
std::cout << "[[" << v[start->second + start->first*3] << ",";          ++start;
std::cout         << v[start->second + start->first*3] << "],\n[";      ++start;
std::cout         << v[start->second + start->first*3] << ",";          ++start;
std::cout         << v[start->second + start->first*3] << "]]";         ++start;

std::cout << "\n\n";

std::cout << "The transpose looks like: ";
dsl::transpose(std::begin(v), std::end(v), 3); // 3 is the number of columns
dsl::print(v,"\n");

start = std::begin(two_by_two);
// To access an element at index (i,j) in the array, use j+i*m, where m is the number of rows
std::cout << "Two by two submatrix of elements (1:2,1:2) \n";
std::cout << "[[" << v[start->second + start->first*3] << ",";          ++start;
std::cout         << v[start->second + start->first*3] << "],\n[";      ++start;
std::cout         << v[start->second + start->first*3] << ",";          ++start;
std::cout         << v[start->second + start->first*3] << "]]";         ++start;

std::cout << "\n\n";

return 0;
}
```

Should produce output

```
{1,2,3,4,5,6,7,8,9}
Two by two submatrix of elements (1:2,1:2)
[[1,2],
[4,5]]

The transpose looks like: {1,4,7,2,5,8,3,6,9}
Two by two submatrix of elements (1:2,1:2)
[[1,4],
[2,5]]
```

6.1.3.87 size_t desalvo_standard_library::two_by_two_map (const std::vector< short > & *v*, const std::vector< std::vector< short >> & *possibles*)

Take a vector of vectors, and ... I don't remember

6.1.3.88 template<typename URNG = std::mt19937> size_t desalvo_standard_library::uniform_size_t (size_t *a*, size_t *b*, URNG & *gen* = `generator_32`) `[inline]`

Returns a random index

Parameters

a	is the lower bound
b	is the upper bound

Template Parameters

gen	is the random generator, by default 32-bit

Returns

a random index between a and b.

6.1.3.89 template<typename InputIterator , typename OutputIterator > OutputIterator desalvo_standard_library::unique_copy_nonconsecutive (InputIterator *start*, InputIterator *stop*, OutputIterator *output*)

copies unique elements from one list to another, does NOT assume the list is sorted. CANNOT be used on pointer types at all.

Template Parameters

InputIterator	is any input iterator type
OutputIterator	is any output iterator type

Parameters

start	refers to first element in range of values to copy
stop	refers to one after last element in range of values to copy
output	refers to first element in container that will receive copied elements

```
#include "desalvo/std_cout.h"
#include "desalvo/numerical.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Throw in some digits of pi
std::vector<int> v {3,1,4,1,5,9,2,6,5,3,5,8,9,7,9,3,2,3};
std::cout << v << std::endl;

// unique out the collection in order
auto it = dsl::unique_copy_nonconsecutive(std::begin(v), std::end(v),
      std::begin(v));

// erase remaining elements so new size of vector is the list without any duplicates, but still in order
v.erase(it,std::end(v));

dsl::print(v,"\n");

// unique out digits which are multiples of 2, i.e., all even numbers are deemed equivalent
it = dsl::unique_copy_nonconsecutive(std::begin(v), std::end(v), std::begin(
      v), [](int a, int b)->bool { return dsl::gcd(a,b)%2==0;});

// erase remaining elements so new size of vector is the list without any duplicates, but still in order
v.erase(it,std::end(v));

dsl::print(v,"\n");

return 0;
}
```

Should produce output

```
{3,1,4,1,5,9,2,6,5,3,5,8,9,7,9,3,2,3}
{3,1,4,5,9,2,6,8,7}
{3,1,4,5,9,7}
```

6.1.3.90 template<typename InputIterator , typename OutputIterator , typename BinaryPredicate > OutputIterator desalvo_standard_library::unique_copy_nonconsecutive (InputIterator *start,* InputIterator *stop,* OutputIterator *output,* BinaryPredicate *bin_op*)

copies unique elements from one list to another, does NOT assume the list is sorted

Template Parameters

InputIterator	is any input iterator type
OutputIterator	is any output iterator type
BinaryPredicate	is any class with a binary predicate function defined

Parameters

start	refers to first element in range of values to copy
stop	refers to one after last element in range of values to copy
output	refers to first element in container that will receive copied elements
bin_op	refers to any function object with appropriately defined binary predicate

```
#include "desalvo/std_cout.h"
#include "desalvo/numerical.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Throw in some digits of pi
std::vector<int> v {3,1,4,1,5,9,2,6,5,3,5,8,9,7,9,3,2,3};
std::cout << v << std::endl;

// unique out the collection in order
auto it = dsl::unique_copy_nonconsecutive(std::begin(v), std::end(v),
      std::begin(v));

// erase remaining elements so new size of vector is the list without any duplicates, but still in order
v.erase(it,std::end(v));

dsl::print(v,"\n");

// unique out digits which are multiples of 2, i.e., all even numbers are deemed equivalent
it = dsl::unique_copy_nonconsecutive(std::begin(v), std::end(v), std::begin(
      v), [](int a, int b)->bool { return dsl::gcd(a,b)%2==0;});

// erase remaining elements so new size of vector is the list without any duplicates, but still in order
v.erase(it,std::end(v));

dsl::print(v,"\n");

return 0;
}
```

Should produce output

```
{3,1,4,1,5,9,2,6,5,3,5,8,9,7,9,3,2,3}
{3,1,4,5,9,2,6,8,7}
{3,1,4,5,9,7}
```

6.1.3.91 std::vector< std::vector<short> > desalvo_standard_library::unique_multiset_subsets (short *n,* short *k*)

Return all subsets of [n] = {1,2,...,n} of size k, i.e., {{1,2,...,k},{1,2,...,k-1,k+1},...,{1,2,...,k-1,n},...{n-k+1,...,n}}. Currently uses inefficient algorithm of generating all first using multiset_subsets and then deleting duplicates.

Parameters

n	is the set of values
k	is the subset size

Returns

all subsets of size k from [n]

```
#include "desalvo/numerical.h"
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

auto s0 = dsl::unique_multiset_subsets(3,0);
auto s1 = dsl::unique_multiset_subsets(3,1);
auto s2 = dsl::unique_multiset_subsets(3,2);
auto s3 = dsl::unique_multiset_subsets(3,3);

dsl::print(s0,"\n");
dsl::print(s1,"\n");
dsl::print(s2,"\n");
dsl::print(s3,"\n");

return 0;
}
```

Should produce output

```
{}
{{1},{2},{3}}
{{1,1},{1,2},{1,3},{2,2},{2,3},{3,3}}
{{1,1,1},{1,1,2},{1,1,3},{1,2,2},{1,2,3},{1,3,3},{2,2,2},{2,2,3},{2,3,3},{3,3,3}}
```

6.2 matlab Namespace Reference

functionality designed to mimic Matlab notation

6.2.1 Detailed Description

functionality designed to mimic Matlab notation Whenever an algorithm is made which happens to have the exact same input/output structure as a Matlab routine, it is placed in this namespace in order to facilitate its use, familiarity with the large amount of user base of Matlab, and encourage further development.

Chapter 7

Class Documentation

7.1 desalvo_standard_library::ArithmeticProgression< T > Class Template Reference

Sequence generator for an arithmetic progression {a, a+r, a+2r, ...}.

`#include <numerical.h>`

Public Member Functions

- ArithmeticProgression (T input_offset, T input_multiple)
- T operator() ()

7.1.1 Detailed Description

template<typename T = size_t>class desalvo_standard_library::ArithmeticProgression< T >

Sequence generator for an arithmetic progression {a, a+r, a+2r, ...}.

Template Parameters

T	is the underlying data type

```
#include "desalvo/numerical.h"
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

dsl::ArithmeticProgression<int> p(3, 7); // 3+7k for k=0,1,2,...
dsl::ArithmeticProgression<int> p2(1, 2); // 1+2k for k=0,1,2,...
dsl::ArithmeticProgression<int> p3(0, 3); // 0+3k for k=0,1,2,...

std::vector<int> v(10);
std::vector<int> v2(10);
std::vector<int> v3(10);

std::generate(std::begin(v), std::end(v), p);
std::generate(std::begin(v2), std::end(v2), p2);
std::generate(std::begin(v3), std::end(v3), p3);

dsl::print(v,"\n");
dsl::print(v2,"\n");
dsl::print(v3,"\n");

return 0;
}
```

Should produce output

```
{3,10,17,24,31,38,45,52,59,66}
{1,3,5,7,9,11,13,15,17,19}
{0,3,6,9,12,15,18,21,24,27}
```

7.1.2 Constructor & Destructor Documentation

7.1.2.1 template<typename T = size_t> desalvo_standard_library::ArithmeticProgression< T >::ArithmeticProgression (T *input_offset*, T *input_multiple*) [inline]

Constructs an arithmetic progression with starting value and multiple value

Parameters

input_offset	is the initial value of the sequence
input_multiple	is the multiples to add to each successive number

7.1.3 Member Function Documentation

7.1.3.1 template<typename T = size_t> T desalvo_standard_library::ArithmeticProgression< T >::operator() ( ) [inline]

get next value in the sequence

Returns

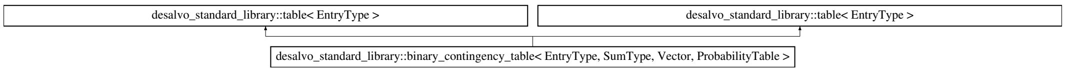
next value in the sequence

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

- DeSalvo Standard Library/desalvo/numerical.h

7.2 desalvo_standard_library::binary_contingency_table< EntryType, SumType, Vector, ProbabilityTable > Class Template Reference

Inheritance diagram for desalvo_standard_library::binary_contingency_table< EntryType, SumType, Vector, ProbabilityTable >:



Public Member Functions

- **binary_contingency_table** (size_t m, size_t n)
- **binary_contingency_table** (size_t m, size_t n)

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

- DeSalvo Standard Library/desalvo/contingency_tables.h
- DeSalvo Standard Library/desalvo/Recycle/contingency_tables.h

7.3 desalvo_standard_library::binary_contingency_table_generator< BoolType, SumType, VectorSumType, ProbabilityTable > Class Template Reference

Public Member Functions

- **binary_contingency_table_generator** (const VectorSumType &row_sums, const VectorSumType &column_sums)
- **binary_contingency_table_generator** (VectorSumType &&row_sums, VectorSumType &&column_sums)
- **binary_contingency_table_generator** (const VectorSumType &row_sums, const VectorSumType &column_sums)
- **binary_contingency_table_generator** (VectorSumType &&row_sums, VectorSumType &&column_sums)

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

- DeSalvo Standard Library/desalvo/contingency_tables.h
- DeSalvo Standard Library/desalvo/Recycle/contingency_tables.h

7.4 desalvo_standard_library::binary_contingency_table_set< EntryType, SumType, VectorSumType, seq, ProbabilityTableType > Class Template Reference

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

- DeSalvo Standard Library/desalvo/contingency_tables.h

7.5 desalvo_standard_library::binary_contingency_table_set< EntryType, SumType, VectorSumType, store::bidirectional, ProbabilityTableType > Class Template Reference

Inheritance diagram for desalvo_standard_library::binary_contingency_table_set< EntryType, SumType, VectorSumType, store::bidirectional, ProbabilityTableType >:

Public Member Functions

- binary_contingency_table_set (const VectorSumType &rowsums, const VectorSumType &columnsums)
- **binary_contingency_table_set** (VectorSumType &&rowsums, VectorSumType &&columnsums)
- V first_in_sequence () const
- V last_in_sequence () const
- bool next_in_sequence (V &v) const
- bool previous_in_sequence (V &v) const
- binary_contingency_table_set (const VectorSumType &rowsums, const VectorSumType &columnsums)
- **binary_contingency_table_set** (VectorSumType &&rowsums, VectorSumType &&columnsums)
- V first_in_sequence () const
- V last_in_sequence () const
- bool next_in_sequence (V &v) const
- bool previous_in_sequence (V &v) const

7.5.1 Constructor & Destructor Documentation

7.5.1.1 template<typename EntryType , typename SumType , typename VectorSumType , typename ProbabilityTableType > desalvo_standard_library::binary_contingency_table_set< EntryType, SumType, VectorSumType, store::bidirectional, ProbabilityTableType >::binary_contingency_table_set (const VectorSumType & *rowsums*, const VectorSumType & *columnsums*) [inline]

Initializes permutation to have size n, computes the first and last elements in the sequence, and stores the entire sequence.

Parameters

input_n	is the initial size of the permutation.

7.5.1.2 template<typename EntryType , typename SumType , typename VectorSumType , typename ProbabilityTableType > desalvo_standard_library::binary_contingency_table_set< EntryType, SumType, VectorSumType, store::bidirectional, ProbabilityTableType >::binary_contingency_table_set (const VectorSumType & *rowsums*, const VectorSumType & *columnsums*) [inline]

Initializes permutation to have size n, computes the first and last elements in the sequence, and stores the entire sequence.

Parameters

input_n	is the initial size of the permutation.

7.5.2 Member Function Documentation

7.5.2.1 template<typename EntryType , typename SumType , typename VectorSumType , typename ProbabilityTableType > V desalvo_standard_library::binary_contingency_table_set< EntryType, SumType, VectorSumType, store::bidirectional, ProbabilityTableType >::first_in_sequence ( ) const [inline]

Compute the first instance of a permutation with given restrictions in lexicographic ordering

Returns

the first permutation in lexicographic ordering with the given restrictions

7.5.2.2 template<typename EntryType , typename SumType , typename VectorSumType , typename ProbabilityTableType > V desalvo_standard_library::binary_contingency_table_set< EntryType, SumType, VectorSumType, store::bidirectional, ProbabilityTableType >::first_in_sequence ( ) const [inline]

Compute the first instance of a permutation with given restrictions in lexicographic ordering

Returns

the first permutation in lexicographic ordering with the given restrictions

7.5.2.3 template<typename EntryType , typename SumType , typename VectorSumType , typename ProbabilityTableType > V desalvo_standard_library::binary_contingency_table_set< EntryType, SumType, VectorSumType, store::bidirectional, ProbabilityTableType >::last_in_sequence ( ) const [inline]

Compute the last instance of a permutation with given restrictions in lexicographic ordering

Returns

the last permutation in lexicographic ordering with the given restrictions

7.5.2.4 template<typename EntryType , typename SumType , typename VectorSumType , typename ProbabilityTableType > V desalvo_standard_library::binary_contingency_table_set< EntryType, SumType, VectorSumType, store::bidirectional, ProbabilityTableType >::last_in_sequence ( ) const [inline]

Compute the last instance of a permutation with given restrictions in lexicographic ordering

Returns

the last permutation in lexicographic ordering with the given restrictions

7.5.2.5 template<typename EntryType , typename SumType , typename VectorSumType , typename ProbabilityTableType > bool desalvo_standard_library::binary_contingency_table_set< EntryType, SumType, VectorSumType, store::bidirectional, ProbabilityTableType >::next_in_sequence (V & *v*) const [inline]

Given a current state, updates the input to the next state, returns false if next state restarts the sequence.

Parameters

v	is the input state, which is updated to the next state

Returns

whether or not the sequence restarted

7.5.2.6 template<typename EntryType , typename SumType , typename VectorSumType , typename ProbabilityTableType > bool desalvo_standard_library::binary_contingency_table_set< EntryType, SumType, VectorSumType, store::bidirectional, ProbabilityTableType >::next_in_sequence (V & *v*) const [inline]

Given a current state, updates the input to the next state, returns false if next state restarts the sequence.

Parameters

v	is the input state, which is updated to the next state

Returns

whether or not the sequence restarted

7.5.2.7 template< typename EntryType , typename SumType , typename VectorSumType , typename ProbabilityTableType > bool desalvo_standard_library::binary_contingency_table_set< EntryType, SumType, VectorSumType, store::bidirectional, ProbabilityTableType >::previous_in_sequence (V & *v*) const [inline]

Given a current state, updates the input to the previous state, returns false if next state restarts the sequence.

Parameters

v	is the input state, which is updated to the previous state

Returns

whether or not the sequence restarted

7.5.2.8 template< typename EntryType , typename SumType , typename VectorSumType , typename ProbabilityTableType > bool desalvo_standard_library::binary_contingency_table_set< EntryType, SumType, VectorSumType, store::bidirectional, ProbabilityTableType >::previous_in_sequence (V & *v*) const [inline]

Given a current state, updates the input to the previous state, returns false if next state restarts the sequence.

Parameters

v	is the input state, which is updated to the previous state

Returns

whether or not the sequence restarted

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

- DeSalvo Standard Library/desalvo/contingency_tables.h
- DeSalvo Standard Library/desalvo/Recycle/contingency_tables.h

7.6 desalvo_standard_library::binary_integer Class Reference

Stores an binary_integer value using bits.

`#include <binary_integer.h>`

Classes

- class binary_integer_string

Public Member Functions

- binary_integer ()

- binary_integer (long long int a)
- binary_integer (const std::string &of_digits)
- binary_integer & operator+= (const binary_integer &rhs)
- binary_integer & operator-= (const binary_integer &rhs)
- binary_integer & operator*= (const binary_integer &rhs)
- binary_integer & operator++ ()
- binary_integer operator++ (int unused)
- binary_integer & operator-- ()
- binary_integer operator-- (int unused)
- binary_integer operator- () const
- binary_integer operator+ () const
- binary_integer & operator&= (const binary_integer &rhs)
- binary_integer & operator|= (const binary_integer &rhs)
- binary_integer & operator^= (const binary_integer &rhs)
- binary_integer & operator<<= (const binary_integer &rhs)
- binary_integer & operator>>= (const binary_integer &rhs)
- binary_integer operator~ () const
- bool operator< (const binary_integer &rhs) const
- bool operator== (const binary_integer &rhs) const
- void print_as_int () const
- void print_as_bits () const
- binary_integer abs () const
- long long int to_llint () const

Friends

- std::ostream & **operator<<** (std::ostream &out, const binary_integer::binary_integer_string &a)
- std::ostream & **operator<<** (std::ostream &out, const binary_integer &a)
- std::istream & **operator>>** (std::istream &in, binary_integer &a)

7.6.1 Detailed Description

Stores an binary_integer value using bits.

This class is designed to mimic the int data type

7.6.2 Constructor & Destructor Documentation

7.6.2.1 desalvo_standard_library::binary_integer::binary_integer ( ) [inline]

Initialize to 0

7.6.2.2 desalvo_standard_library::binary_integer::binary_integer (long long int *a*)

Initialize using variable of type long long int

Parameters

a	is the initial value

7.6.2.3 desalvo_standard_library::binary_integer::binary_integer (const std::string & *of_digits*)

Input using a string of digits

Parameters

of_digits	contains the digits in the usual notation, with a leading minus sign for negative numbers

7.6.3 Member Function Documentation

7.6.3.1 binary_integer desalvo_standard_library::binary_integer::abs ( ) const

Converts the binary_integer to its absolute value, simply by changing the sign bit

Returns

a new binary_integer with the absolute value of the origin

7.6.3.2 binary_integer & desalvo_standard_library::binary_integer::operator&= (const binary_integer & *rhs*)

Performs bit-wise & on all bits, filling in 0s when necessary

Parameters

rhs	is the right hand side of a &= b;

Returns

a reference for chaining

7.6.3.3 binary_integer & desalvo_standard_library::binary_integer::operator*= (const binary_integer & *rhs*)

Multiplies two binary_integer together, a *= b, using the elementary school algorithm.

Parameters

rhs	is the right hand side of the equation a *= b

Returns

a reference to a in a *= b;

7.6.3.4 binary_integer desalvo_standard_library::binary_integer::operator+ ( ) const

Make a copy

Returns

a copy of the value

7.6.3.5 binary_integer & desalvo_standard_library::binary_integer::operator++ ( )

Increment by 1

Returns

a reference to the incremented value

7.6.3.6 binary_integer desalvo_standard_library::binary_integer::operator++ (int *unused*)

Increment by 1

Returns

the value before the increment

7.6.3.7 binary_integer & desalvo_standard_library::binary_integer::operator+= (const binary_integer & *rhs*)

Add two binary_integers together, a += b

Parameters

rhs	is the right hand side of the equation a += b;

Returns

a reference to a in a+=b;

7.6.3.8 binary_integer desalvo_standard_library::binary_integer::operator- ( ) const

Compute the negative

Returns

the negative of the value

7.6.3.9 binary_integer & desalvo_standard_library::binary_integer::operator-- ( )

Decrement by 1

Returns

a reference to the decremented value

7.6.3.10 binary_integer desalvo_standard_library::binary_integer::operator-- (int *unused*)

Decrement by 1

Returns

the value before the decrement

7.6.3.11 binary_integer & desalvo_standard_library::binary_integer::operator-= (const binary_integer & *rhs*)

Subtract two binary_integers together, a -= b, store the result in a

Parameters

rhs	is the right hand side of the equation a -= b;

Returns

a reference to a in a-=b;

7.6.3.12 bool desalvo_standard_library::binary_integer::operator< (const binary_integer & *rhs*) const

Compares a < b for binary_integer types

Parameters

rhs	is the right hand sidein a < b

Returns

true if a < b, i.e., if a is strictly less than b

7.6.3.13 binary_integer & desalvo_standard_library::binary_integer::operator<<= (const binary_integer & *rhs*)

Bit shift up operator, transforms 1011 –> 101100 when shifted by 2

Parameters

rhs	is the amount to shift by, which should be small and positive

Returns

a reference to the object being shifted for chaining.

7.6.3.14 bool desalvo_standard_library::binary_integer::operator== (const binary_integer & *rhs*) const

Compares for equality, a == b

Parameters

rhs	is the right hand side of a == b

Returns

true if a equals b, false otherwise

7.6.3.15 binary_integer & desalvo_standard_library::binary_integer::operator>>= (const binary_integer & *rhs*)

Bit shift down operator, transforms 1011 –> 10 when shifted down by 2

Parameters

rhs	is the amount to shift by, which should be small and positive

Returns

a reference to the object being shifted for chaining.

7.6.3.16 binary_integer & desalvo_standard_library::binary_integer::operator^= (const binary_integer & *rhs*)

Performs bit-wise xor ^ operation

Parameters

rhs	is the right hand side argument in a ^= b;

Returns

a reference for chaining

7.6.3.17 binary_integer & desalvo_standard_library::binary_integer::operator\|= (const binary_integer & *rhs*)

Performs bit-wise or | operation

Parameters

rhs	is the right hand side argument in a |= b;

Returns

a reference for chaining

7.6.3.18 binary_integer desalvo_standard_library::binary_integer::operator~ ( ) const

Negate all bits, including signed bit

Returns

the negation of the binary_integer

7.6.3.19 void desalvo_standard_library::binary_integer::print_as_bits ( ) const

Prints the number in terms of its bits

7.6.3.20 void desalvo_standard_library::binary_integer::print_as_int ( ) const

Prints the number as a long long int

7.6.3.21 long long int desalvo_standard_library::binary_integer::to_llint ( ) const

Convert to a long long int value

Returns

the binary_integer as a long long int

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

- DeSalvo Standard Library/desalvo/binary_integer.h
- DeSalvo Standard Library/desalvo/binary_integer.cpp

7.7 desalvo_standard_library::binary_integer::binary_integer_string Class Reference

Public Member Functions

- **binary_integer_string** (int n)
- **binary_integer_string** (std::string dig)
- **binary_integer_string** (const binary_integer &b)
- binary_integer_string **MultiplyBy2** ()
- binary_integer_string **operator+** (const binary_integer_string &i)
- binary_integer_string **operator+** (int i)
- binary_integer_string **operator+** (unsigned int i)
- binary_integer_string **operator+** (unsigned long long i)

Friends

- binary_integer_string **operator+** (int j, const binary_integer_string &i)
- std::ostream & **operator**<< (std::ostream &out, const binary_integer_string &a)
- std::ostream & **operator**<< (std::ostream &out, const binary_integer &a)
- std::ostream & **operator**<< (std::ostream &out, const binary_integer_string &a)

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

- DeSalvo Standard Library/desalvo/binary_integer.cpp

7.8 binary_integerString Class Reference

Arbitrary Precision class for binary_integers.

7.8.1 Detailed Description

Arbitrary Precision class for binary_integers.

I'm finally going to just make an arbitrary precision class for binary_integers.

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

- DeSalvo Standard Library/desalvo/binary_integer.cpp

7.9 desalvo_standard_library::sudoku< ValueType >::block_iterator Class Reference

```
#include <sudoku.h>
```

Public Member Functions

- **block_iterator** (size_t i)

7.9.1 Detailed Description

template<typename ValueType = short>class desalvo_standard_library::sudoku< ValueType >::block_iterator

iterator that iterates through elements in a given block.

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

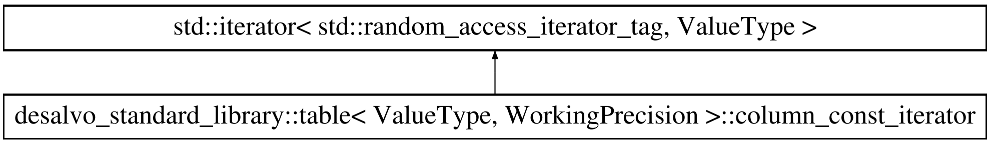
- DeSalvo Standard Library/desalvo/sudoku.h

7.10 desalvo_standard_library::table< ValueType, WorkingPrecision >::column_const_iterator Class Reference

Random Access const_iterator for columns.

```
#include <table.h>
```

Inheritance diagram for desalvo_standard_library::table< ValueType, WorkingPrecision >::column_const_iterator:



Public Member Functions

- column_const_iterator (const table *initial_mat=nullptr, int initial_row=0, int initial_col=0)
- column_const_iterator (const column_const_iterator &other)
- void swap (column_const_iterator &other)
- column_const_iterator & operator= (column_const_iterator to_copy)
- column_const_iterator & operator++ ()
- column_const_iterator operator++ (int)
- column_const_iterator & operator+= (int r)
- column_const_iterator operator+ (int r) const
- column_const_iterator & operator-- ()
- column_const_iterator operator-- (int)
- column_const_iterator & operator-= (int r)
- column_const_iterator operator- (int r) const
- int operator- (const column_const_iterator &p2) const
- ValueType & operator* () const
- ValueType & operator-> () const
- ValueType & operator[] (int n) const

Friends

- bool operator== (const table::column_const_iterator &lhs, const table::column_const_iterator &rhs)
- bool operator< (const table::column_const_iterator &lhs, const table::column_const_iterator &rhs)

7.10.1 Detailed Description

template<typename ValueType = double, typename WorkingPrecision = long double>class desalvo_standard_library::table< ValueType, WorkingPrecision >::column_const_iterator

Random Access const_iterator for columns.

This class is designed to be a RANDOM ACCESS const_iterator for a given column of the entry.

The row is declared const so that it cannot be modified once set. The col is modified along with the pointer so that it is easier to keep track of bounds.

```
#include <iostream>
#include <vector>
#include "std_cout.h"
#include "desalvo/table.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Make a vector of values
std::vector<int> v(100);

// initialize values to 0,1,2,...,99
std::iota(std::begin(v), std::end(v), 0);

// Initialize 10x10 table using 10 numbers for each row from v
dsl::table<int> t(10,10,std::begin(v));

// print out the table of values first
std::cout << "t: \n" << t << std::endl << std::endl;

// Iterate through every other row, squaring each element in each row
for(auto i = 0; i<10; i += 2) {
// Square each value
std::for_each(t.begin_row(i), t.end_row(i), [](int& a) { a *= a; });
}

std::cout << "Printing every third column ... \n";
// Iterate through every third column, print it out
for(auto i = 0; i<10; i += 3) {
// Print every other row
std::cout << "{";
std::for_each(t.cbegin_column(i), t.cend_column(i), [](int a) { std::cout << a<<","; });
std::cout << "}\n\n";
}

// Sort each row...no good reason, just want to demonstrate that the iterators work even for algorithms
       which require random access iterators.  Note that begin_row and end_row return objects, not raw pointers.  Use
       begin_row_raw and end_row_raw to obtain the raw pointer types.
for(auto i = 0;i<10;++i)
std::sort(t.begin_row(i), t.end_row(i));

std::cout << "After all that, sort elements in each row, and print t again:\n";
std::cout << t << std::endl;

return 0;
}
```

Should produce output

```
t:
{{0,1,2,3,4,5,6,7,8,9},
{10,11,12,13,14,15,16,17,18,19},
{20,21,22,23,24,25,26,27,28,29},
{30,31,32,33,34,35,36,37,38,39},
{40,41,42,43,44,45,46,47,48,49},
{50,51,52,53,54,55,56,57,58,59},
{60,61,62,63,64,65,66,67,68,69},
{70,71,72,73,74,75,76,77,78,79},
{80,81,82,83,84,85,86,87,88,89},
{90,91,92,93,94,95,96,97,98,99}}

Printing every third column ...
{0,10,400,30,1600,50,3600,70,6400,90,}

{9,13,529,33,1849,53,3969,73,6889,93,}

{36,16,676,36,2116,56,4356,76,7396,96,}
```

```
{81,19,841,39,2401,59,4761,79,7921,99,}

After all that, sort elements in each row, and print t again:
{{0,1,4,9,16,25,36,49,64,81},
{10,11,12,13,14,15,16,17,18,19},
{400,441,484,529,576,625,676,729,784,841},
{30,31,32,33,34,35,36,37,38,39},
{1600,1681,1764,1849,1936,2025,2116,2209,2304,2401},
{50,51,52,53,54,55,56,57,58,59},
{3600,3721,3844,3969,4096,4225,4356,4489,4624,4761},
{70,71,72,73,74,75,76,77,78,79},
{6400,6561,6724,6889,7056,7225,7396,7569,7744,7921},
{90,91,92,93,94,95,96,97,98,99}}
```

7.10.2 Constructor & Destructor Documentation

7.10.2.1 template<typename ValueType = double, typename WorkingPrecision = long double> **desalvo_standard_library::table< ValueType, WorkingPrecision >::column_const_iterator::column_const_iterator** (const **table** * *initial_mat* = nullptr, int *initial_row* = 0, int *initial_col* = 0) [inline]

Construct by table object

Parameters

T	is the table object with data
r	is the row number.
col	is the column

7.10.2.2 template<typename ValueType = double, typename WorkingPrecision = long double> **desalvo_standard_library::table< ValueType, WorkingPrecision >::column_const_iterator::column_const_iterator** (const **column_const_iterator** & *other*) [inline]

copy constructor, no heap memory managed so simple copy of all members, not really necessary to write explicitly since compiler would generate this for us.

Parameters

other	

7.10.3 Member Function Documentation

7.10.3.1 template<typename ValueType = double, typename WorkingPrecision = long double> ValueType& **desalvo_standard_library::table< ValueType, WorkingPrecision >::column_const_iterator::operator*** ( ) const [inline]

Dereference operator

Returns

the value the const_iterator points to.

7.10.3.2 template<typename ValueType = double, typename WorkingPrecision = long double> **column_const_iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::column_const_iterator::operator+** (int *r*) const [inline]

Increment operator.

Parameters

r	is the increment, can be negative

Returns

a new iterator referring to the original element plus the input offset.

7.10.3.3 template<typename ValueType = double, typename WorkingPrecision = long double> column_const_iterator& desalvo_standard_library::table< ValueType, WorkingPrecision >::column_const_iterator::operator++ ( ) [inline]

Standard prefix ++ operator

Returns

reference to self after the increment.

7.10.3.4 template<typename ValueType = double, typename WorkingPrecision = long double> column_const_iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::column_const_iterator::operator++ (int) [inline]

Postfix ++ operator

7.10.3.5 template<typename ValueType = double, typename WorkingPrecision = long double> column_const_iterator& desalvo_standard_library::table< ValueType, WorkingPrecision >::column_const_iterator::operator+= (int *r*) [inline]

Increment equals operator, updated value of iterator

Parameters

r	is the increment, can be negative

Returns

a reference to the newly increment iterator for chaining.

7.10.3.6 template<typename ValueType = double, typename WorkingPrecision = long double> column_const_iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::column_const_iterator::operator- (int *r*) const [inline]

Decrement operator.

Parameters

r	is the decrement, can be negative

Returns

a new iterator referring to the original element minus the input offset.

7.10.3.7 template<typename ValueType = double, typename WorkingPrecision = long double> int desalvo_standard_library::table< ValueType, WorkingPrecision >::column_const_iterator::operator- (const column_const_iterator & *p2*) const [inline]

Take the difference between two iterators of the same type, ∗this-p2

Parameters

p2	is the rhs of ∗this-p2

Returns

the number of elements that must be transversed in order to get from ∗this to p2; can be negative.

7.10.3.8 template<typename ValueType = double, typename WorkingPrecision = long double> column_const_iterator& desalvo_standard_library::table< ValueType, WorkingPrecision >::column_const_iterator::operator-- ( ) [inline]

Standard prefix – operator

Returns

reference to self after the increment.

7.10.3.9 template<typename ValueType = double, typename WorkingPrecision = long double> column_const_iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::column_const_iterator::operator-- (int) [inline]

Postfix – operator

7.10.3.10 template<typename ValueType = double, typename WorkingPrecision = long double> column_const_iterator& desalvo_standard_library::table< ValueType, WorkingPrecision >::column_const_iterator::operator-= (int *r*) [inline]

Decrement equals operator, updated value of iterator

Parameters

r	is the decrement, can be negative

Returns

a reference to the newly decremented iterator for chaining.

7.10.3.11 template<typename ValueType = double, typename WorkingPrecision = long double> ValueType& desalvo_standard_library::table< ValueType, WorkingPrecision >::column_const_iterator::operator-> ( ) const [inline]

Dereference operator

Returns

the value the const_iterator points to. Equivalent to operator∗ by dereferencing twice.

7.10.3.12 template< typename ValueType = double, typename WorkingPrecision = long double> column_const_iterator& desalvo_standard_library::table< ValueType, WorkingPrecision >::column_const_iterator::operator= (column_const_iterator *to_copy*) [inline]

Assignment operator, copies over using Copy & Swap idiom

Parameters

to_copy	is the existing object to copy from

Returns

a reference to the newly updated object, for chaining.

7.10.3.13 template< typename ValueType = double, typename WorkingPrecision = long double> ValueType& desalvo_standard_library::table< ValueType, WorkingPrecision >::column_const_iterator::operator[] (int *n*) const [inline]

Random access operator, in order to mimic pointer.

Parameters

n	is the offset, can be negative

Returns

a reference to the element referred to by the iterator and offset.

7.10.3.14 template< typename ValueType = double, typename WorkingPrecision = long double> void desalvo_standard_library::table< ValueType, WorkingPrecision >::column_const_iterator::swap (column_const_iterator & *other*) [inline]

swaps two iterators

Parameters

other	is the other iterator

7.10.4 Friends And Related Function Documentation

7.10.4.1 template< typename ValueType = double, typename WorkingPrecision = long double> bool operator< (const table::column_const_iterator & *lhs*, const table::column_const_iterator & *rhs*) [friend]

Tests for const_iterators in same row but strictly smaller column

Parameters

t	is the other const_iterator

Returns

true if const_iterators are equivalent

7.10.4.2 template<typename ValueType = double, typename WorkingPrecision = long double> bool operator== (const table::column_const_iterator & *lhs*, const table::column_const_iterator & *rhs*) [friend]

Tests for const_iterators in two equivalent positions

Parameters

t	is the other const_iterator

Returns

true if const_iterators are equivalent

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

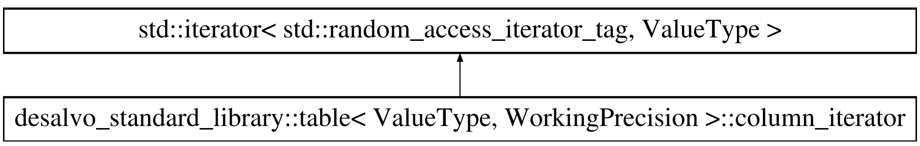
- DeSalvo Standard Library/desalvo/table.h

7.11 desalvo_standard_library::table< ValueType, WorkingPrecision >::column_iterator Class Reference

Random Access Iterator for columns.

#include <table.h>

Inheritance diagram for desalvo_standard_library::table< ValueType, WorkingPrecision >::column_iterator:



Public Member Functions

- column_iterator (table *initial_mat, int initial_row=0, int initial_col=0)
- column_iterator (const column_iterator &other)
- column_iterator (column_iterator &&other)
- void swap (column_iterator &other)
- column_iterator & operator= (column_iterator to_copy)
- column_iterator & operator++ ()
- column_iterator operator++ (int)
- column_iterator & **operator+=** (int r)
- column_iterator **operator+** (int r)
- column_iterator & operator-- ()
- column_iterator operator-- (int)
- column_iterator & **operator-=** (int r)
- column_iterator **operator-** (int r)
- int operator- (const column_iterator &p2) const
- const ValueType & operator* () const
- ValueType & operator* ()
- const ValueType & operator-> () const
- ValueType & operator-> ()
- ValueType & **operator[]** (int n)
- const ValueType & **operator[]** (int n) const

Friends

- bool operator== (const table::column_iterator &lhs, const table::column_iterator &rhs)
- bool operator< (const table::column_iterator &lhs, const table::column_iterator &rhs)

7.11.1 Detailed Description

template<typename ValueType = double, typename WorkingPrecision = long double>class desalvo_standard_library::table< ValueType, WorkingPrecision >::column_iterator

Random Access Iterator for columns.

This class is designed to be a RANDOM ACCESS ITERATOR for a given column of the entry.

The row is declared const so that it cannot be modified once set. The col is modified along with the pointer so that it is easier to keep track of bounds.

```
#include <iostream>
#include <vector>
#include "std_cout.h"
#include "desalvo/table.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Make a vector of values
std::vector<int> v(100);

// initialize values to 0,1,2,...,99
std::iota(std::begin(v), std::end(v), 0);

// Initialize 10x10 table using 10 numbers for each row from v
dsl::table<int> t(10,10,std::begin(v));

// print out the table of values first
std::cout << "t: \n" << t << std::endl << std::endl;

// Iterate through every fourth column, squaring each element
for(auto j = 0; j<10; j += 4) {
// Square each value
std::for_each(t.begin_column(j), t.end_column(j), [](int& a) { a *= a; });
}

std::cout << "Printing every other column ... \n";
// Iterate through every other row, print it out
for(auto i = 0; i<10; i += 2) {
// Print every other row
std::cout << "{";
std::for_each(t.cbegin_row(i), t.cend_row(i), [](int a) { std::cout << a<<","; });
std::cout << "}\n\n";
}

// Sort each column...no good reason, just want to demonstrate that the iterators work even for algorithms
       which require random access iterators.  Note that begin_column and end_column return objects, not raw
       pointers, since raw pointers will not observe the desired behavior for the ROW-MAJOR table format.
for(auto i = 0;i<10;++i)
std::sort(t.begin_column(i), t.end_column(i));

std::cout << "After all that, sort elements in each column, and print t again:\n";
std::cout << t << std::endl;

return 0;
}
```

Should produce output

```
t:
{{0,1,2,3,4,5,6,7,8,9},
{10,11,12,13,14,15,16,17,18,19},
{20,21,22,23,24,25,26,27,28,29},
{30,31,32,33,34,35,36,37,38,39},
{40,41,42,43,44,45,46,47,48,49},
{50,51,52,53,54,55,56,57,58,59},
{60,61,62,63,64,65,66,67,68,69},
{70,71,72,73,74,75,76,77,78,79},
{80,81,82,83,84,85,86,87,88,89},
```

```
{90,91,92,93,94,95,96,97,98,99}}

Printing every other column ...
{0,1,2,3,16,5,6,7,64,9,}

{400,21,22,23,576,25,26,27,784,29,}

{1600,41,42,43,1936,45,46,47,2304,49,}

{3600,61,62,63,4096,65,66,67,4624,69,}

{6400,81,82,83,7056,85,86,87,7744,89,}

After all that, sort elements in each column, and print t again:
{{0,1,2,3,16,5,6,7,64,9},
{100,11,12,13,196,15,16,17,324,19},
{400,21,22,23,576,25,26,27,784,29},
{900,31,32,33,1156,35,36,37,1444,39},
{1600,41,42,43,1936,45,46,47,2304,49},
{2500,51,52,53,2916,55,56,57,3364,59},
{3600,61,62,63,4096,65,66,67,4624,69},
{4900,71,72,73,5476,75,76,77,6084,79},
{6400,81,82,83,7056,85,86,87,7744,89},
{8100,91,92,93,8836,95,96,97,9604,99}}
```

7.11.2 Constructor & Destructor Documentation

7.11.2.1 template<typename ValueType = double, typename WorkingPrecision = long double> desalvo_standard_library::table< ValueType, WorkingPrecision >::column_iterator::column_iterator (table ∗ *initial_mat*, int *initial_row* = 0, int *initial_col* = 0) [inline]

Construct by table object

Parameters

T	is the table object with data
r	is the row number.
col	is the column

7.11.2.2 template<typename ValueType = double, typename WorkingPrecision = long double> desalvo_standard_library::table< ValueType, WorkingPrecision >::column_iterator::column_iterator (const column_iterator & *other*) [inline]

Copy constructor, follows default

Parameters

other	is the existing object from which to copy from

7.11.2.3 template<typename ValueType = double, typename WorkingPrecision = long double> desalvo_standard_library::table< ValueType, WorkingPrecision >::column_iterator::column_iterator (column_iterator && *other*) [inline]

Copy constructor, follows default

Parameters

other	is the existing object from which to copy from

7.11.3 Member Function Documentation

7.11.3.1 template< typename ValueType = double, typename WorkingPrecision = long double > const ValueType& desalvo_standard_library::table< ValueType, WorkingPrecision >::column_iterator::operator* ( ) const [inline]

Dereference operator

Returns

the value the iterator points to.

7.11.3.2 template< typename ValueType = double, typename WorkingPrecision = long double > ValueType& desalvo_standard_library::table< ValueType, WorkingPrecision >::column_iterator::operator* ( ) [inline]

Dereference operator

Returns

the value the iterator points to.

7.11.3.3 template< typename ValueType = double, typename WorkingPrecision = long double > column_iterator& desalvo_standard_library::table< ValueType, WorkingPrecision >::column_iterator::operator++ ( ) [inline]

Standard prefix ++ operator

Returns

reference to self after the increment.

7.11.3.4 template< typename ValueType = double, typename WorkingPrecision = long double > column_iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::column_iterator::operator++ (int) [inline]

Postfix ++ operator

7.11.3.5 template< typename ValueType = double, typename WorkingPrecision = long double > int desalvo_standard_library::table< ValueType, WorkingPrecision >::column_iterator::operator- (const column_iterator & *p2*) const [inline]

Take the difference between two iterators of the same type, ∗this-p2

Parameters

p2	is the rhs of ∗this-p2

Returns

the number of elements that must be transversed in order to get from *this to p2; can be negative.

7.11.3.6 template<typename ValueType = double, typename WorkingPrecision = long double> column_iterator& desalvo_standard_library::table< ValueType, WorkingPrecision >::column_iterator::operator-- ( ) [inline]

Standard prefix – operator

Returns

reference to self after the increment.

7.11.3.7 template<typename ValueType = double, typename WorkingPrecision = long double> column_iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::column_iterator::operator-- (int) [inline]

Postfix – operator

7.11.3.8 template<typename ValueType = double, typename WorkingPrecision = long double> const ValueType& desalvo_standard_library::table< ValueType, WorkingPrecision >::column_iterator::operator-> ( ) const [inline]

Dereference operator

Returns

the value the iterator points to. Equivalent to operator* by dereferencing twice.

7.11.3.9 template<typename ValueType = double, typename WorkingPrecision = long double> ValueType& desalvo_standard_library::table< ValueType, WorkingPrecision >::column_iterator::operator-> ( ) [inline]

Dereference operator

Returns

the value the iterator points to. Equivalent to operator* by dereferencing twice.

7.11.3.10 template<typename ValueType = double, typename WorkingPrecision = long double> column_iterator& desalvo_standard_library::table< ValueType, WorkingPrecision >::column_iterator::operator= (column_iterator *to_copy*) [inline]

Assignment operator, copies over using Copy & Swap idiom

Parameters

to_copy	is the existing object to copy from

Returns

a reference to the newly updated object, for chaining.

7.11.3.11 template<typename ValueType = double, typename WorkingPrecision = long double> void desalvo_standard_library::table< ValueType, WorkingPrecision >::column_iterator::swap (column_iterator & *other*) [inline]

Swaps two iterators, even those referring to different tables

Parameters

other	is the other iterator.

7.11.4 Friends And Related Function Documentation

7.11.4.1 template<typename ValueType = double, typename WorkingPrecision = long double> bool operator< (const table::column_iterator & *lhs*, const table::column_iterator & *rhs*) [friend]

Tests for iterators in same row but strictly smaller column

Parameters

t	is the other iterator

Returns

true if iterators are equivalent

7.11.4.2 template<typename ValueType = double, typename WorkingPrecision = long double> bool operator== (const table::column_iterator & *lhs*, const table::column_iterator & *rhs*) [friend]

Tests for iterators in two equivalent positions

Parameters

t	is the other iterator

Returns

true if iterators are equivalent

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

- DeSalvo Standard Library/desalvo/table.h

7.12 combinatorial_structure< Derived, ObjectType, ComponentType, Collection > Class Template Reference

Classes

- class object

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

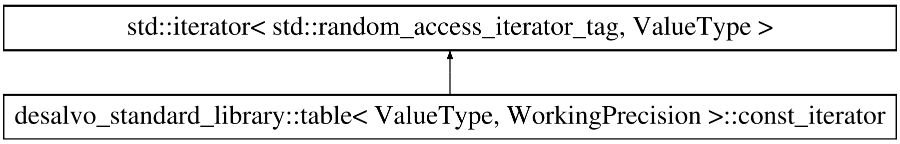
- DeSalvo Standard Library/desalvo/combinatorial_structure.h

7.13 desalvo_standard_library::table< ValueType, WorkingPrecision >::const_iterator Class Reference

random access const iterator for all entries in table

```
#include <table.h>
```

Inheritance diagram for desalvo_standard_library::table< ValueType, WorkingPrecision >::const_iterator:



Public Member Functions

- const_iterator (const table *initial_mat=nullptr, int initial_row=0, int initial_col=0)
- const_iterator (const const_iterator &other)
- void swap (const_iterator &other)
- const_iterator & operator= (const_iterator to_copy)
- const_iterator & operator++ ()
- const_iterator operator++ (int unused)
- const_iterator & operator+= (int col)
- const_iterator operator+ (int col) const
- const_iterator & operator-- ()
- const_iterator operator-- (int unused)
- const_iterator & operator-= (int col)
- const_iterator operator- (int col) const
- const_iterator::iterator::difference_type operator- (const const_iterator &p2) const
- ValueType & operator* () const
- ValueType & operator-> () const
- ValueType & operator[] (int n) const

Friends

- bool operator== (const table::const_iterator &lhs, const table::const_iterator &rhs)
- bool operator< (const table::const_iterator &lhs, const table::const_iterator &rhs)

7.13.1 Detailed Description

template< typename ValueType = double, typename WorkingPrecision = long double >class desalvo_standard_library::table< ValueType, WorkingPrecision >::const_iterator

random access const iterator for all entries in table

This iterator treats the entries in the table as a 1D array of values. It is useful if the same operation or transformation needs to be applied to all entries in the table.

In principle it is faster to use the raw pointers to iterate through all entries, but this provides a unified object interface, especially when working with column iterators, which must be objects since the operations do not translate over literally.

```
#include <iostream>
#include <vector>
#include "std_cout.h"
#include "desalvo/table.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Create 10x10 table, default initialized values (i.e., 0 for int)
dsl::table<int> t(10,10);

// Very simple linear congruential engine
int A = 16807;
int C = 127;
int value = 1;

// initialize values using custom linear congruential engine
for(auto& i : t)
i = (value = ( (A*value)%C));

std::cout << "t: \n" << t << std::endl << std::endl;

// Sort the entire set of values
std::sort(t.begin(), t.end());

std::cout << "Sort all elements, and print t again:\n";
std::cout << t << std::endl << std::endl;

// Check for duplicates
auto start = t.cbegin();
auto stop = t.cend();

// check is initialized one before start, which now points to second element
auto checker = start++;

unsigned int number_of_duplicates = 0;

// loop through all elements, essentially treating table as 1D array
while(start != stop) {

// check for equailty, increment
if(*checker++ == *start++)
++number_of_duplicates;
}

std::cout << "The number of duplicate elements is: " << number_of_duplicates << std::endl << std::endl;

// I get 0 duplicates, which is not very random!
// This is an example of the birthday problem.  At each new element created, there is a chance it will
       match an existing element.  Assuming perfect randomness, the probability that there are no duplicates in 100
       values of 127 possible values is given by
// (1)(1-1/127)(1-2/127)...(1-99/127)
// Let's do that calculation quickly.
double x = 1.;

for(size_t i=1;i<100;++i)
x *= 1.-i/127.;

std::cout << "The probability of having 0 elements if they were in fact generated perfectly randomly is: "
      << x << ".\n\n";

std::cout << "Do you need further anecdotal evidence to suggest that linear congruential generators are not
       great randomizers?" << std::endl;

return 0;
}
```

Should produce output

```
t:
{{43,71,5,88,101,25,59,124,125,41},
{112,117,78,52,77,9,6,4,45,30},
{20,98,23,100,109,115,119,37,67,87},
{58,81,54,36,24,16,53,120,80,11},
{92,19,55,79,95,21,14,94,105,70},
{89,17,96,64,85,99,66,44,114,76},
{93,62,126,84,56,122,39,26,102,68},
{3,2,86,15,10,49,75,50,118,121},
{123,82,97,107,29,104,27,18,12,8},
{90,60,40,69,46,73,91,103,111,74}}

Sort all elements, and print t again:
{{2,3,4,5,6,8,9,10,11,12},
{14,15,16,17,18,19,20,21,23,24},
```

```
{25,26,27,29,30,36,37,39,40,41},
{43,44,45,46,49,50,52,53,54,55},
{56,58,59,60,62,64,66,67,68,69},
{70,71,73,74,75,76,77,78,79,80},
{81,82,84,85,86,87,88,89,90,91},
{92,93,94,95,96,97,98,99,100,101},
{102,103,104,105,107,109,111,112,114,115},
{117,118,119,120,121,122,123,124,125,126}}

The number of duplicate elements is: 0

The probability of having 0 elements if they were in fact generated perfectly randomly is: 1.15238e-25.

Do you need further anecdotal evidence to suggest that linear congruential generators are not great
      randomizers?
```

7.13.2 Constructor & Destructor Documentation

7.13.2.1 template<typename ValueType = double, typename WorkingPrecision = long double> desalvo_standard_library::table< ValueType, WorkingPrecision >::const_iterator::const_iterator (const table * *initial_mat* = nullptr, int *initial_row* = 0, int *initial_col* = 0) [inline]

Constructor with parameters with default-initialized parameters so also includes the default constructor. Initializes iterator with a table and an initial location on the table

Parameters

initial_mat	is the table to iterate through
initial_row	is the row number.
initial_col	is the column number.

7.13.2.2 template<typename ValueType = double, typename WorkingPrecision = long double> desalvo_standard_library::table< ValueType, WorkingPrecision >::const_iterator::const_iterator (const const_iterator & *other*) [inline]

copy constructor

Parameters

other	is the existing interator to copy from

7.13.3 Member Function Documentation

7.13.3.1 template<typename ValueType = double, typename WorkingPrecision = long double> ValueType& desalvo_standard_library::table< ValueType, WorkingPrecision >::const_iterator::operator* ( ) const [inline]

Accesses the entry directly relative to mat, so even if mat assumes new memory or location it still points to relatively the same location

Returns

reference to (row,column)-th entry in mat

7.13.3.2 template<typename ValueType = double, typename WorkingPrecision = long double> const_iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::const_iterator::operator+ (int *col*) const [inline]

increment operator by integer value

Parameters

col	is the number of entries to increment

Returns

a copy of the iterator pointing to the newly incremented value

7.13.3.3 template< typename ValueType = double, typename WorkingPrecision = long double> const_iterator& desalvo_standard_library::table< ValueType, WorkingPrecision >::const_iterator::operator++ ( ) [inline]

Standard prefix ++ operator

Returns

reference to self after the increment.

7.13.3.4 template< typename ValueType = double, typename WorkingPrecision = long double> const_iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::const_iterator::operator++ (int *unused*) [inline]

Postfix ++ operator

Parameters

unused	is an unused parameter to distinguish from the prefix operator++.

7.13.3.5 template< typename ValueType = double, typename WorkingPrecision = long double> const_iterator& desalvo_standard_library::table< ValueType, WorkingPrecision >::const_iterator::operator+= (int *col*) [inline]

increment operator by integer value

Parameters

col	is the number of entries to increment

Returns

a reference to the iterator for chaining.

7.13.3.6 template< typename ValueType = double, typename WorkingPrecision = long double> const_iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::const_iterator::operator- (int *col*) const [inline]

decrement operator by integer value

Parameters

col	is the number of entries to decrement

Returns

a copy of the iterator pointing to the newly decremented value

7.13.3.7 template<typename ValueType = double, typename WorkingPrecision = long double> const_iterator::iterator::difference_type desalvo_standard_library::table< ValueType, WorkingPrecision >::const_iterator::operator- (const const_iterator & *p2*) const [inline]

Take the difference between two iterators of the same type, ∗this-p2

Parameters

p2	is the rhs of ∗this-p2

Returns

the number of elements that must be transversed in order to get from ∗this to p2; can be negative.

7.13.3.8 template<typename ValueType = double, typename WorkingPrecision = long double> const_iterator& desalvo_standard_library::table< ValueType, WorkingPrecision >::const_iterator::operator-- ( ) [inline]

Standard prefix – operator

Returns

reference to self after the decrement.

7.13.3.9 template<typename ValueType = double, typename WorkingPrecision = long double> const_iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::const_iterator::operator-- (int *unused*) [inline]

Postfix – operator

Parameters

unused	is an unused parameter to distinguish from the prefix operator–

Returns

a copy of the iterator to the original position before the decrement

7.13.3.10 template<typename ValueType = double, typename WorkingPrecision = long double> const_iterator& desalvo_standard_library::table< ValueType, WorkingPrecision >::const_iterator::operator-= (int *col*) [inline]

decrement operator by integer value

Parameters

col	is the number of entries to decrement

Returns

a reference to the iterator for chaining.

7.13.3.11 template<typename ValueType = double, typename WorkingPrecision = long double> ValueType& desalvo_standard_library::table< ValueType, WorkingPrecision >::const_iterator::operator-> ( ) const [inline]

Accesses the entry directly relative to mat, so even if mat assumes new memory or location it still points to relatively the same location

Returns

reference to (row,column)-th entry in mat

7.13.3.12 template<typename ValueType = double, typename WorkingPrecision = long double> const_iterator& desalvo_standard_library::table< ValueType, WorkingPrecision >::const_iterator::operator= (const_iterator *to_copy*) [inline]

Assignment operator, grabs values and copies over

Parameters

to_copy	is the object from which to copy over the values.

7.13.3.13 template<typename ValueType = double, typename WorkingPrecision = long double> ValueType& desalvo_standard_library::table< ValueType, WorkingPrecision >::const_iterator::operator[] (int *n*) const [inline]

Accesses the entry n past the current point

Returns

reference to (row,column+n)-th entry in mat

7.13.3.14 template<typename ValueType = double, typename WorkingPrecision = long double> void desalvo_standard_library::table< ValueType, WorkingPrecision >::const_iterator::swap (const_iterator & *other*) [inline]

swaps two iterators

Parameters

other	is the object to swap with

7.13.4 Friends And Related Function Documentation

7.13.4.1 template<typename ValueType = double, typename WorkingPrecision = long double> bool operator< (const table::const_iterator & *lhs*, const table::const_iterator & *rhs*) [friend]

Tests for iterators in two equivalent positions

Parameters

t	is the other iterator

Returns

true if iterators are equivalent

7.13.4.2 template< typename ValueType = double, typename WorkingPrecision = long double> bool operator== (const table::const_iterator & *lhs*, const table::const_iterator & *rhs*) [friend]

Tests for iterators in two equivalent positions

Parameters

t	is the other iterator

Returns

true if iterators are equivalent

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

- DeSalvo Standard Library/desalvo/table.h

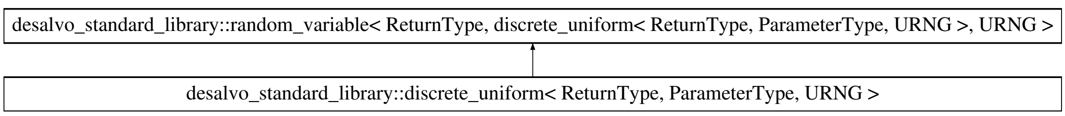
7.14 desalvo_standard_library::decomposable_structure< iparcs > Class Template Reference

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

- DeSalvo Standard Library/desalvo/iparcs.h

7.15 desalvo_standard_library::discrete_uniform< ReturnType, ParameterType, URNG > Class Template Reference

Inheritance diagram for desalvo_standard_library::discrete_uniform< ReturnType, ParameterType, URNG >:



Public Member Functions

- discrete_uniform (ParameterType a, ParameterType b)
- ReturnType operator() (URNG &gen=generator_64)
- template<typename F = double>
  F mean ()

7.15.1 Constructor & Destructor Documentation

7.15.1.1 template<typename ReturnType = int, typename ParameterType = ReturnType, typename URNG = std::mt19937_64> desalvo_standard_library::discrete_uniform< ReturnType, ParameterType, URNG >::discrete_uniform (ParameterType *a*, ParameterType *b*) [inline]

Constructs a random variable for random values in {a,a+1,...,b-1,b}

Parameters

a	is the lower bound
b	is the upper bound

7.15.2 Member Function Documentation

7.15.2.1 template<typename ReturnType = int, typename ParameterType = ReturnType, typename URNG = std::mt19937_64> template<typename F = double> F desalvo_standard_library::discrete_uniform< ReturnType, ParameterType, URNG >::mean ( ) [inline]

Returns the expected value of the random variable ReturnType must have operator+(ReturnType, ReturnType) defined, and the return type of operator+ must be castable to F

Returns

(lower+upper)/2

7.15.2.2 template<typename ReturnType = int, typename ParameterType = ReturnType, typename URNG = std::mt19937_64> ReturnType desalvo_standard_library::discrete_uniform< ReturnType, ParameterType, URNG >::operator() (URNG & *gen* = generator_64) [inline]

Generates random value from distribution using the std distributions.

Parameters

gen	is the random number generator, by default 64-bit.

Returns

random element

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

- DeSalvo Standard Library/desalvo/statistics.h

7.16 DiscreteUniform Class Reference

Uniform over set of elements {a,a+1,...,b-1,b}.

`#include <statistics.h>`

7.16.1 Detailed Description

Uniform over set of elements {a,a+1,...,b-1,b}.

Inherits from RandomVariable so as long as operator()(URNG& gen) is overloaded to return a random value we can invoke its member functions.

Store a lower and upper bound denoting the smallest and largest values capable of being generated.

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

- DeSalvo Standard Library/desalvo/statistics.h

7.17 desalvo_standard_library::DivisibleBy Class Reference

Creates function objects which check for divisibility.

```
#include <numerical.h>
```

Public Member Functions

- DivisibleBy (unsigned long in)
- bool operator() (unsigned long x)

7.17.1 Detailed Description

Creates function objects which check for divisibility.

```
#include "desalvo/numerical.h"
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Form matrix [[1,2,3],[4,5,6],[7,8,9]] using a row-major 1D vector
std::vector<int> v {1,2,3,4,5,6,7,8,9,10};
std::vector<int> v2(10);

auto it = std::copy_if( std::begin(v), std::end(v), std::begin(v2),
      dsl::DivisibleBy(3) );

// optional erase, makes output cleaner
v2.erase(it, std::end(v2));


dsl::print(v,"\n");
dsl::print(v2,"\n");


return 0;
}
```

Should produce output

```
{1,2,3,4,5,6,7,8,9,10}
{3,6,9}
```

7.17.2 Constructor & Destructor Documentation

7.17.2.1 desalvo_standard_library::DivisibleBy::DivisibleBy (unsigned long *in*) [inline]

Construct a function object with a given divisibility condition

Parameters

in	is the divisibility value

7.17.3 Member Function Documentation

7.17.3.1 bool desalvo_standard_library::DivisibleBy::operator() (unsigned long *x*) [inline]

Checks if input is divisible by n

Parameters

x	is the input to check for divisibility

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

- DeSalvo Standard Library/desalvo/numerical.h

7.18 desalvo_standard_library::file< type > Class Template Reference

Partially specialized for input and output.

`#include <file.h>`

7.18.1 Detailed Description

template<file_type type>class desalvo_standard_library::file< type >

Partially specialized for input and output.

This class is designed to be extended in various directions.

1. input – handle for input from a file
2. output – handle for outputting to a file
3. console – handle for std::cin and std::cout

The plan is to make the following extensions

1. SmartOutput – auto detects if file already exists when writing and changes name
2. AppendOutput – Appends to already existing file
3. IncrementalOutput – Used when needing to write many files with same name but different indices e.g., Data-_n_is_2, Data_n_is_4, Data_n_is_8, etc.

Example 1:

```
std::string filename = "/Users/stephendesalvo/Documents/";

dsl::file< output > f(filename + "file.txt");
dsl::file< input  > f_in(filename + "file_to_read_from.txt");

dsl::file< console > screen;

//f.write("Hi there!", "\n");
f << "Hi there!";

std::string text;

f_in.getline(text);

std::cout << text << std::endl;

screen << text << std::endl;
```

```
screen.write(text);

screen.read(text);
screen.write(text, "\n");
screen.write(text, std::endl);

screen.getline(text);
screen.write(text);

screen << "Same as before \n";
screen << "Can do endl as well" << std::endl;
```

Example 2: Produces a file which contains random numbers between 1 and 6.

```
// Program to generate random dice rolls
#include "desalvo/std_cout.h"
#include "desalvo/file.h"

#include <random>
#include "desalvo/statistics.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Local file path as std::string for easy concatenation
std::string prefix = "/Users/stephendesalvo/Documents/";

dsl::file< dsl::file_type::output > file(prefix + "dice_data.txt");

std::uniform_int_distribution<int> unif(1,6);

file << "{";
for(auto i=0;i<9999;++i)
file << unif(dsl::generator_64) << ",";

file << unif(dsl::generator_64) << "}";

return 0;
}
```

Example 3: Generate a matrix and store it in a file

```
// Program to generate rectangular version of Hilbert matrix
#include "std_cin.h"
#include "std_cout.h"
#include "file.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Local file path as std::string for easy concatenation
std::string prefix = "/Users/stephendesalvo/Documents/";

// Earlier, we made a file with numbers between 1 and 6 contained in dice_data.txt

dsl::file< dsl::file_type::output > make_matlab_matrix(prefix + "
      matlab_matrix.txt");

size_t m = 10;
size_t n = 5;

make_matlab_matrix << "[";
for(size_t i=0;i<m;++i) {
for(size_t j=0;j<n-1;++j) {
make_matlab_matrix << 1./(1.+i+j) << ",";
}
make_matlab_matrix << 1./(1.+i+(n-1.)) << ";";
}
make_matlab_matrix << "]";

return 0;
}
```

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

- DeSalvo Standard Library/desalvo/file.h

7.19 desalvo_standard_library::file< file_type::console > Class Template Reference

Public Member Functions

- file (std::fstream::openmode additional_modes=std::fstream::out, size_t output_precision=10)
- template<typename T >
  file & **read** (T &&p)
- operator bool ()
- template<typename T >
  file & **operator>>** (T &&p)
- template<typename T >
  file & **operator<<** (T &&t)
- template<typename T >
  file & **write** (T &&t)
- template<typename Streamsize = size_t>
  file & **ignore** (Streamsize n=1, int delim=EOF)
- template<typename T , typename String = std::string>
  file & **write** (T &&t, String &&ending=std::string(""))
- template<typename T >
  file & **write** (T &&t, manip1 ending)
- template<typename T >
  file & **read** (T &t)
- file & operator<< (manip1 fp)
- file & operator<< (manip2 fp)
- file & operator<< (manip3 fp)
- file & **operator<<** (const std::vector< int > &v)
- template<typename Streamsize >
  file< file_type::console > & ignore (Streamsize n, int delim)
- template<typename T >
  file< file_type::console > & operator>> (T &&p)
- template<typename T >
  file< file_type::console > & operator<< (T &&t)
- template<typename T >
  file< file_type::console > & write (T &&t, manip1 fp)
- template<typename T >
  file< file_type::console > & read (T &t)

Friends

- template<typename String >
  bool **getline** (file &fin, String &s)

7.19.1 Constructor & Destructor Documentation

7.19.1.1 desalvo_standard_library::file< file_type::console >::file (std::fstream::openmode *additional_modes* = `std::fstream::out`, size_t *output_precision* = `10`)

Constructor via filename and various modes and options

Parameters

filename	is the full filepath of the file
additional_-modes	is a collection of options for formatting output
output_precision	is for outputting numerical floating point values

```
#include "desalvo/std_cout.h"
#include "desalvo/file.h"

#include <random>

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Local file path as std::string for easy concatenation
std::string prefix = "/Users/stephendesalvo/Documents/";

dsl::file< dsl::file_type::output > file(prefix + "data.txt");

file << 3;
file << '.';
file << 1<<4<<1<<5<<9<<2<<6<<5;

// too slow?
std::string more_digits = "35897932384626433832795028841971693993751058209";

file << more_digits;

return 0;
}
```

Doesn't produce any output to the console, but in the file data.txt we should see

```
3.14159265358979323846264338327950288419716939937510582 09
```

7.19.2 Member Function Documentation

7.19.2.1 template<typename Streamsize > file<file_type::console>& desalvo_standard_library::file< file_type::console >::ignore (Streamsize *n*, int *delim*)

Mimics the ignore function in the istream library; that is, it ignores the next n characters or until the delim character is reached, discarding the delim character.

Template Parameters

Streamsize	is any unsigned integer type

Parameters

n	is the number of characters to ignore
delim	is a char with which to stop reading characters

```
// Program to generate rectangular version of Hilbert matrix
#include "std_cin.h"
#include "std_cout.h"
#include "file.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

dsl::file<dsl::file_type::console> console;

console << "Hello World!\n";

std::vector<std::string> v(5);

console << "Go ahead, write 5 words of text:" << std::endl;
console >> v[0];
console >> v[1];
console >> v[2];
console >> v[3];
console >> v[4];

// There is an extra return carriage we would very must like to ignore
console.ignore();

console << "Let me make sure I got that: \n";
console << v << std::endl;

console << "Now please write 5 lines of text:" << std::endl;
dsl::getline(console, v[0]);
dsl::getline(console, v[1]);
dsl::getline(console, v[2]);
dsl::getline(console, v[3]);
dsl::getline(console, v[4]);

console << "Let me make sure I got that: \n";
console << v << std::endl;


return 0;
}
```

Sample input/output

```
Hello World!
Go ahead, write 5 words of text:
Now is the winter of
Let me make sure I got that:
{Now,is,the,winter,of}
Now please write 5 lines of text:
Now is the winter of our discontent
Made glorious summer by this sun of York;
And all the clouds that lour'd upon our house
In the deep bosom of the ocean buried.
Now are our brows bound with victorious wreaths;
Let me make sure I got that:
{Now is the winter of our discontent,Made glorious summer by this sun of York;,And all the clouds that lou
      'd upon our house,In the deep bosom of the ocean buried.,Now are our brows bound with victorious wre
```

7.19.2.2 desalvo_standard_library::file< file_type::console >::operator bool ( )

Makes the file convertable to bool for use in conditional expressions

7.19.2.3 file< file_type::console > & desalvo_standard_library::file< file_type::console >::operator<< (manip1 *fp*)

Output for manipulators

Parameters

fp	is an argument like std::endl

Returns

a reference to the file object for chaining

```
#include "desalvo/std_cout.h"
#include "desalvo/file.h"
#include <iomanip>

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

dsl::file< dsl::file_type::console > console;

console << std::setprecision(20);

console << 3.14159265358979323846264338832 << std::endl;

console << std::setw(20) << std::left;

console << 3 << std::setw(20) << 12 << std::setw(10) << 4 << std::setw(5) << 9987;

return 0;
}
```

Should produce output

```
3.141592653589793116
3                   12                  4         9987
```

7.19.2.4 file< file_type::console > & desalvo_standard_library::file< file_type::console >::operator<< (manip2 *fp*)

Output for manipulators

Parameters

fp	is an argument which manipulates the underlying file stream

Returns

a reference to the file object for chaining

7.19.2.5 file< file_type::console > & desalvo_standard_library::file< file_type::console >::operator<< (manip3 *fp*)

Output for manipulators

Parameters

fp	is an argument which manipulates the underlying file stream

Returns

a reference to the file object for chaining

```
#include "desalvo/std_cout.h"
#include "desalvo/file.h"
#include <iomanip>

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

dsl::file< dsl::file_type::console > console;

console << std::setprecision(20);

console << 3.14159265358979323846264338832 << std::endl;

console << std::setw(20) << std::left;
```

```
console << 3 << std::setw(20) << 12 << std::setw(10) << 4 << std::setw(5) << 9987;

return 0;
}
```

Should produce output

```
3.141592653589793116
3                   12                  4         9987
```

7.19.2.6 template<typename T > file<file_type::console>& desalvo_standard_library::file< file_type::console >::operator<< (T && *t*)

Outputs argument to underlying file stream

Template Parameters

t	is the input for the file stream

Returns

a reference to the file object for chaining

```
// Program to generate rectangular version of Hilbert matrix
#include "std_cin.h"
#include "std_cout.h"
#include "file.h"


namespace dsl = desalvo_standard_library;

// custom class to handle 2D data point, in order to demonstrate how to integrate custom code with existing
       library.
class Point2D {
// output operator: std::cout << pt << std::endl;
friend std::ostream& operator<<(std::ostream& out, const Point2D& pt) {
return out << "(" << pt.x << "," << pt.y << ")";
}

// input operator: std::cin >> pt;
friend std::istream& operator>>(std::istream& in, Point2D& pt) {
char unused;

// format is: (x,y)
// More generally: (char)x(char)y(char)
in >> unused;   // (
in >> pt.x;     //  x
in >> unused;   //   ,
in >> pt.y;     //    y
in >> unused;   //     )

return in;
}
public:
// Initialize value of point
Point2D(double input_x, double input_y) : x(input_x), y(input_y) { };

// Default constructor, calls more general constructor, new C++11.
Point2D() : Point2D(0,0) { };
private:
double x, y;
};

int main(int argc, const char * argv[]) {

dsl::file<dsl::file_type::console> console;

std::pair<int, int> my_pair;
std::vector<int> v;
std::set<double> v2;
std::set<int> v3;
std::multiset<int> v4;
std::list<Point2D> v5;
std::valarray<double> v6;
std::array<Point2D,4> v7;

console << "Input values in the same format as the default dsl output: \n";
console << "Insert pair values: ";
```

```
console >> my_pair;
console << "Insert vector of ints: ";
console >> v;
console << "Insert set of doubles: ";
console >> v2;
console << "Insert set of ints: ";
console >> v3;
console << "Insert multiset of ints: ";
console >> v4;
console << "Insert list of Point2Ds: ";
console >> v5;
console << "Insert valarray of doubles: ";
console >> v6;
console << "Insert array of 4 Point2Ds: ";
console >> v7;

console << "pair stored as: " << my_pair << std::endl;
console << "vector of ints: " << v << std::endl;
console << "set of doubles: " << v2 << std::endl;
console << "set of ints: " << v3 << std::endl;
console << "multiset of ints: " << v4 << std::endl;
console << "list of Point2Ds: " << v5 << std::endl;
console << "valarray of doubles: " << v6 << std::endl;
console << "array of Point2D: " << v7 << std::endl;


return 0;
}
```

Should produce output like

```
Input values in the same format as the default dsl output:
Insert pair values: {1,2}
Insert vector of ints: {5,4,2,1}
Insert set of doubles: {1.1,1.1,2.2,2.4,3.7}
Insert set of ints: {1,1,1,1,2,2,2,3,3,4,5}
Insert multiset of ints: {1,1,1,1,2,2,2,3,3,4,5}
Insert list of Point2Ds: {(1,2),(3.4,5.4),(0,0)}
Insert valarray of doubles: {1.1,2.2,3.3,4.4,5.5,6.6}
Insert array of 4 Point2Ds: {(0,0),(-1,1),(-1,-1),(1,-1)}
pair stored as: {1,2}
vector of ints: {5,4,2,1}
set of doubles: {1.1,2.2,2.4,3.7}
set of ints: {1,2,3,4,5}
multiset of ints: {1,1,1,1,2,2,2,3,3,4,5}
list of Point2Ds: {(1,2),(3.4,5.4),(0,0)}
valarray of doubles: {1.1,2.2,3.3,4.4,5.5,6.6}
array of Point2D: {(0,0),(-1,1),(-1,-1),(1,-1)}
```

7.19.2.7 template<typename T > file<file_type::console>& desalvo_standard_library::file< file_type::console >::operator>> (T && *p*)

Passes along the input to the stream

Template Parameters

T	is the type of object

Parameters

p	is the stream input

Returns

reference to the File object for chaining

```
// Program to generate rectangular version of Hilbert matrix
#include "std_cin.h"
#include "std_cout.h"
#include "file.h"


namespace dsl = desalvo_standard_library;

// custom class to handle 2D data point, in order to demonstrate how to integrate custom code with existing
```

```
        library.
class Point2D {
// output operator: std::cout << pt << std::endl;
friend std::ostream& operator<<(std::ostream& out, const Point2D& pt) {
return out << "(" << pt.x << "," << pt.y << ")";
}

// input operator: std::cin >> pt;
friend std::istream& operator>>(std::istream& in, Point2D& pt) {
char unused;

// format is: (x,y)
// More generally: (char)x(char)y(char)
in >> unused;   // (
in >> pt.x;     //  x
in >> unused;   //   ,
in >> pt.y;     //    y
in >> unused;   //     )

return in;
}
public:
// Initialize value of point
Point2D(double input_x, double input_y) : x(input_x), y(input_y) { };

// Default constructor, calls more general constructor, new C++11.
Point2D() : Point2D(0,0) { };
private:
double x, y;
};

int main(int argc, const char * argv[]) {

dsl::file<dsl::file_type::console> console;

std::pair<int, int> my_pair;
std::vector<int> v;
std::set<double> v2;
std::set<int> v3;
std::multiset<int> v4;
std::list<Point2D> v5;
std::valarray<double> v6;
std::array<Point2D,4> v7;

console << "Input values in the same format as the default dsl output: \n";
console << "Insert pair values: ";
console >> my_pair;
console << "Insert vector of ints: ";
console >> v;
console << "Insert set of doubles: ";
console >> v2;
console << "Insert set of ints: ";
console >> v3;
console << "Insert multiset of ints: ";
console >> v4;
console << "Insert list of Point2Ds: ";
console >> v5;
console << "Insert valarray of doubles: ";
console >> v6;
console << "Insert array of 4 Point2Ds: ";
console >> v7;

console << "pair stored as: " << my_pair << std::endl;
console << "vector of ints: " << v << std::endl;
console << "set of doubles: " << v2 << std::endl;
console << "set of ints: " << v3 << std::endl;
console << "multiset of ints: " << v4 << std::endl;
console << "list of Point2Ds: " << v5 << std::endl;
console << "valarray of doubles: " << v6 << std::endl;
console << "array of Point2D: " << v7 << std::endl;


return 0;
}
```

Should produce output like

```
Input values in the same format as the default dsl output:
Insert pair values: {1,2}
Insert vector of ints: {5,4,2,1}
Insert set of doubles: {1.1,1.1,2.2,2.4,3.7}
Insert set of ints: {1,1,1,1,2,2,2,3,3,4,5}
Insert multiset of ints: {1,1,1,1,2,2,2,3,3,4,5}
Insert list of Point2Ds: {(1,2),(3.4,5.4),(0,0)}
Insert valarray of doubles: {1.1,2.2,3.3,4.4,5.5,6.6}
Insert array of 4 Point2Ds: {(0,0),(-1,1),(-1,-1),(1,-1)}
pair stored as: {1,2}
```

```
vector of ints: {5,4,2,1}
set of doubles: {1.1,2.2,2.4,3.7}
set of ints: {1,2,3,4,5}
multiset of ints: {1,1,1,1,2,2,2,3,3,4,5}
list of Point2Ds: {(1,2),(3.4,5.4),(0,0)}
valarray of doubles: {1.1,2.2,3.3,4.4,5.5,6.6}
array of Point2D: {(0,0),(-1,1),(-1,-1),(1,-1)}
```

7.19.2.8 template<typename T > file<file_type::console>& desalvo_standard_library::file< file_type::console >::read (T & *t*)

Loads in next value of type T from stream

Template Parameters

T	must have operator>>(istream&, T&) defined

Parameters

t	is an object reference

Returns

false if file is no longer in valid state

```
// Program to generate rectangular version of Hilbert matrix
#include "std_cin.h"
#include "std_cout.h"
#include "file.h"

namespace dsl = desalvo_standard_library;

// custom class to handle 2D data point, in order to demonstrate how to integrate custom code with existing
       library.
class Point2D {
// output operator: std::cout << pt << std::endl;
friend std::ostream& operator<<(std::ostream& out, const Point2D& pt) {
return out << "(" << pt.x << "," << pt.y << ")";
}

// input operator: std::cin >> pt;
friend std::istream& operator>>(std::istream& in, Point2D& pt) {
char unused;

// format is: (x,y)
// More generally: (char)x(char)y(char)
in >> unused;   // (
in >> pt.x;     //  x
in >> unused;   //   ,
in >> pt.y;     //    y
in >> unused;   //     )

return in;
}
public:
// Initialize value of point
Point2D(double input_x, double input_y) : x(input_x), y(input_y) { };

// Default constructor, calls more general constructor, new C++11.
Point2D() : Point2D(0,0) { };
private:
double x, y;
};

int main(int argc, const char * argv[]) {

dsl::file<dsl::file_type::console> console;

std::pair<int, int> my_pair;
std::vector<int> v;
std::set<double> v2;
std::set<int> v3;
std::multiset<int> v4;
std::list<Point2D> v5;
std::valarray<double> v6;
std::array<Point2D,4> v7;
```

```
console << "Input values in the same format as the default dsl output: \n";
console << "Insert pair values: ";          console.read(my_pair);
console << "Insert vector of ints: ";       console.read(v);
console << "Insert set of doubles: ";       console.read(v2);
console << "Insert set of ints: ";          console.read(v3);
console << "Insert multiset of ints: ";     console.read(v4);
console << "Insert list of Point2Ds: ";     console.read(v5);
console << "Insert valarray of doubles: ";  console.read(v6);
console << "Insert array of 4 Point2Ds: ";  console.read(v7);

console << "pair stored as: ";      console.write(my_pair,std::endl);
console << "vector of ints: ";      console.write(v, std::endl);
console << "set of doubles: ";      console.write(v2,std::endl);
console << "set of ints: ";         console.write(v3,std::endl);
console << "multiset of ints: ";    console.write(v4,std::endl);
console << "list of Point2Ds: ";    console.write(v5,std::endl);
console << "valarray of doubles: "; console.write(v6,std::endl);
console << "array of Point2D: ";    console.write(v7,std::endl);

return 0;
}
```

Should produce output like

```
Input values in the same format as the default dsl output:
Insert pair values: {1,2}
Insert vector of ints: {5,4,2,1}
Insert set of doubles: {1.1,1.1,2.2,5.4,6.5,3.14159}
Insert set of ints: {1,1,1,1,2,2,2,3,3,4,5}
Insert multiset of ints: {1,1,1,1,2,2,2,3,3,4,5}
Insert list of Point2Ds: {(0,0),(-1.1,1),(-2,2)}
Insert valarray of doubles: {1.1,1.2,1.3,1.4,1.5}
Insert array of 4 Point2Ds: {(0,0),(1,1),(2,2),(3,3)}
pair stored as: {1,2}
vector of ints: {5,4,2,1}
set of doubles: {1.1,2.2,3.14159,5.4,6.5}
set of ints: {1,2,3,4,5}
multiset of ints: {1,1,1,1,2,2,2,3,3,4,5}
list of Point2Ds: {(0,0),(-1.1,1),(-2,2)}
valarray of doubles: {1.1,1.2,1.3,1.4,1.5}
array of Point2D: {(0,0),(1,1),(2,2),(3,3)}
```

7.19.2.9 template<typename T > file<file_type::console>& desalvo_standard_library::file< file_type::console >::write (T && *t*, manip1 *fp*)

Outputs argument to underlying file stream

Template Parameters

t	is the input for the file stream

Returns

a reference to the file object for chaining

```
// Program to generate rectangular version of Hilbert matrix
#include "std_cin.h"
#include "std_cout.h"
#include "file.h"


namespace dsl = desalvo_standard_library;

// custom class to handle 2D data point, in order to demonstrate how to integrate custom code with existing
    library.
class Point2D {
// output operator: std::cout << pt << std::endl;
friend std::ostream& operator<<(std::ostream& out, const Point2D& pt) {
return out << "(" << pt.x << "," << pt.y << ")";
}

// input operator: std::cin >> pt;
friend std::istream& operator>>(std::istream& in, Point2D& pt) {
char unused;

// format is: (x,y)
// More generally: (char)x(char)y(char)
```

```
in >> unused;   // (
in >> pt.x;     //  x
in >> unused;   //   ,
in >> pt.y;     //    y
in >> unused;   //     )

return in;
}
public:
// Initialize value of point
Point2D(double input_x, double input_y) : x(input_x), y(input_y) { };

// Default constructor, calls more general constructor, new C++11.
Point2D() : Point2D(0,0) { };
private:
double x, y;
};

int main(int argc, const char * argv[]) {

dsl::file<dsl::file_type::console> console;

std::pair<int, int> my_pair;
std::vector<int> v;
std::set<double> v2;
std::set<int> v3;
std::multiset<int> v4;
std::list<Point2D> v5;
std::valarray<double> v6;
std::array<Point2D,4> v7;

console << "Input values in the same format as the default dsl output: \n";
console << "Insert pair values: ";
console >> my_pair;
console << "Insert vector of ints: ";
console >> v;
console << "Insert set of doubles: ";
console >> v2;
console << "Insert set of ints: ";
console >> v3;
console << "Insert multiset of ints: ";
console >> v4;
console << "Insert list of Point2Ds: ";
console >> v5;
console << "Insert valarray of doubles: ";
console >> v6;
console << "Insert array of 4 Point2Ds: ";
console >> v7;

console << "pair stored as: ";      console.write(my_pair,std::endl);
console << "vector of ints: ";      console.write(v, std::endl);
console << "set of doubles: ";      console.write(v2,std::endl);
console << "set of ints: ";         console.write(v3,std::endl);
console << "multiset of ints: ";    console.write(v4,std::endl);
console << "list of Point2Ds: ";    console.write(v5,std::endl);
console << "valarray of doubles: "; console.write(v6,std::endl);
console << "array of Point2D: ";    console.write(v7,std::endl);

return 0;
}
```

Should produce output like

```
Input values in the same format as the default dsl output:
Insert pair values: {1,2}
Insert vector of ints: {5,4,2,1}
Insert set of doubles: {1.1,1.1,2.2,2.4,3.7}
Insert set of ints: {1,1,1,1,2,2,2,3,3,4,5}
Insert multiset of ints: {1,1,1,1,2,2,2,3,3,4,5}
Insert list of Point2Ds: {(1,2),(3.4,5.4),(0,0)}
Insert valarray of doubles: {1.1,2.2,3.3,4.4,5.5,6.6}
Insert array of 4 Point2Ds: {(0,0),(-1,1),(-1,-1),(1,-1)}
pair stored as: {1,2}
vector of ints: {5,4,2,1}
set of doubles: {1.1,2.2,2.4,3.7}
set of ints: {1,2,3,4,5}
multiset of ints: {1,1,1,1,2,2,2,3,3,4,5}
list of Point2Ds: {(1,2),(3.4,5.4),(0,0)}
valarray of doubles: {1.1,2.2,3.3,4.4,5.5,6.6}
array of Point2D: {(0,0),(-1,1),(-1,-1),(1,-1)}
```

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

- DeSalvo Standard Library/desalvo/file.h

7.20 desalvo_standard_library::file< file_type::input > Class Template Reference

Public Member Functions

- file (const std::string &filename)
- file (file &&other)
- file & operator= (file &&other)
- ∼file ()
- template<typename Streamsize = size_t>
  file & **ignore** (Streamsize n=1, int delim=EOF)
- template<typename T >
  file & **operator>>** (T &p)
- template<typename T >
  file & **read** (T &t)
- template<typename String = std::string>
  bool getline (String &line)
- operator bool ()
- template<typename T >
  file< file_type::input > & operator>> (T &p)
- template<typename T >
  file< file_type::input > & read (T &t)
- template<typename Streamsize >
  file< file_type::input > & ignore (Streamsize n, int delim)

Friends

- bool **getline** (file &fin, std::string &s)

7.20.1 Constructor & Destructor Documentation

7.20.1.1 desalvo_standard_library::file< file_type::input >::file (const std::string & *filename*)

Initialize using filename

Parameters

filename	contains the name of the file

```
// Program to load in random dice rolls
#include "std_cin.h"
#include "std_cout.h"
#include "file.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Local file path as std::string for easy concatenation
std::string prefix = "/Users/stephendesalvo/Documents/";

// Earlier, we made a file with numbers between 1 and 6 contained in dice_data.txt

// Prepare file for loading in values.
dsl::file< dsl::file_type::input > file(prefix + "data_dice.txt");

// We shall store them here
std::vector<int> rolls;

// Grab values.  We don't know how many in advance so by default it calls push_back
file >> rolls;

std::cout << rolls << std::endl;

return 0;
}
```

Should produce output to the console depending on contents of file, e.g.,

```
{1,6,4,3,2,2,3,1,5,6}
```

7.20.1.2 desalvo_standard_library::file< file_type::input >::file (file< file_type::input > && *other*)

Move constructor

Parameters

other	is the expiring file stream

```
// Program to load in random dice rolls
#include "std_cin.h"
#include "std_cout.h"
#include "file.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Local file path as std::string for easy concatenation
std::string prefix = "/Users/stephendesalvo/Documents/";

// Earlier, we made a file with numbers between 1 and 6 contained in dice_data.txt

// Prepare file for loading in values.
dsl::file< dsl::file_type::input > file(prefix + "data_dice.txt");

// Changed my mind, would prefer to use file_pair handle
dsl::file< dsl::file_type::input > file_pair(std::move(file));

// We shall store them here
std::vector<int> rolls;

// Grab values.  We don't know how many in advance so by default it calls push_back
file_pair >> rolls;

std::cout << rolls << std::endl;

return 0;
}
```

Should produce output to the console depending on contents of file, e.g.,

```
{1,6,4,3,2,2,3,1,5,6}
```

7.20.1.3 desalvo_standard_library::file< file_type::input >::~file ( )

Closes out the stream. +1 for RAII!

7.20.2 Member Function Documentation

7.20.2.1 template<typename String = std::string> bool desalvo_standard_library::file< file_type::input >::getline (String & *line*)

Wrapper for the std::getline function

Template Parameters

String	is the type of object to load into

Parameters

line	contains the next line in the file.

Returns

whether the file is in a valid state after the get

```
// Program to load in random dice rolls
#include "std_cin.h"
#include "std_cout.h"
#include "file.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Local file path as std::string for easy concatenation
std::string prefix = "/Users/stephendesalvo/Documents/";

// Earlier, we made a file with numbers between 1 and 6 contained in dice_data.txt

// Prepare file for loading in values.
dsl::file< dsl::file_type::input > file(prefix + "
      data_richard_iii_opening_monologue_short.txt");

std::string line;

while( dsl::getline(file, line) )
std::cout << line << std::endl;

return 0;
}
```

Should produce output

```
Now is the winter of our discontent
Made glorious summer by this sun of York;
And all the clouds that lour'd upon our house
In the deep bosom of the ocean buried.
Now are our brows bound with victorious wreaths;
Our bruised arms hung up for monuments;
Our stern alarums changed to merry meetings,
Our dreadful marches to delightful measures.
Grim-visaged war hath smooth'd his wrinkled front;
And now, instead of mounting barded steeds
To fright the souls of fearful adversaries,
He capers nimbly in a lady's chamber
To the lascivious pleasing of a lute.
```

7.20.2.2 template<typename Streamsize > file<file_type::input>& desalvo_standard_library::file< file_type::input >::ignore (Streamsize *n*, int *delim*)

Mimics the ignore function in the istream library; that is, it ignores the next n characters or until the delim character is reached, discarding the delim character.

Template Parameters

Streamsize	is any unsigned integer type

Parameters

n	is the number of characters to ignore
delim	is a char with which to stop reading characters

7.20.2.3 desalvo_standard_library::file< file_type::input >::operator bool ( )

Makes the file convertable to bool for use in conditional expressions

Returns

false when the stream fails

```
// Program to load in random dice rolls
#include "std_cin.h"
#include "std_cout.h"
#include "file.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Local file path as std::string for easy concatenation
std::string prefix = "/Users/stephendesalvo/Documents/";

// Earlier, we made a file with numbers between 1 and 6 contained in dice_data.txt

// Prepare file for loading in values.
dsl::file< dsl::file_type::input > file(prefix + "
      data_richard_iii_opening_monologue_short.txt");

std::string line;

while( dsl::getline(file, line) )
std::cout << line << std::endl;

return 0;
}
```

Should produce output

```
Now is the winter of our discontent
Made glorious summer by this sun of York;
And all the clouds that lour'd upon our house
In the deep bosom of the ocean buried.
Now are our brows bound with victorious wreaths;
Our bruised arms hung up for monuments;
Our stern alarums changed to merry meetings,
Our dreadful marches to delightful measures.
Grim-visaged war hath smooth'd his wrinkled front;
And now, instead of mounting barded steeds
To fright the souls of fearful adversaries,
He capers nimbly in a lady's chamber
To the lascivious pleasing of a lute.
```

7.20.2.4 file< file_type::input > & desalvo_standard_library::file< file_type::input >::operator= (file< file_type::input > && *other*)

move assignment operator

Parameters

other	is the expiring file stream

Returns

reference to file for chaining

```
// Program to load in random dice rolls
#include "std_cin.h"
#include "std_cout.h"
#include "file.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Local file path as std::string for easy concatenation
std::string prefix = "/Users/stephendesalvo/Documents/";

// Earlier, we made a file with numbers between 1 and 6 contained in dice_data.txt

// Prepare file for loading in values.
dsl::file< dsl::file_type::input > file(prefix + "data_dice.txt");
dsl::file< dsl::file_type::input > file_pair(prefix + "data_pair.txt");

// Changed my mind, would prefer to use file_pair, so safely close data_pair.txt and take over
       data_dice.txt
file_pair = std::move(file);

// We shall store them here
std::vector<int> rolls;

// Grab values.  We don't know how many in advance so by default it calls push_back
file_pair >> rolls;

std::cout << rolls << std::endl;

return 0;
}
```

Should produce output to the console depending on contents of file, e.g.,

```
{1,6,4,3,2,2,3,1,5,6}
```

7.20.2.5 template< typename T > file< file_type::input >& desalvo_standard_library::file< file_type::input >::operator>> (T & *p*)

Passes along the input to the stream, object must have operator>>(istream&,T&) -> istream& defined

Template Parameters

T	is the type of object

Parameters

p	is the stream input

Returns

reference to the File object for chaining

```
// Program to load in random dice rolls
#include "std_cin.h"
#include "std_cout.h"
#include "file.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Local file path as std::string for easy concatenation
std::string prefix = "/Users/stephendesalvo/Documents/";

// Earlier, we made a file with numbers between 1 and 6 contained in dice_data.txt

// Prepare file for loading in values.
dsl::file< dsl::file_type::input > file(prefix + "data_dice.txt");
```

```
// We shall store them here
std::vector<int> rolls;

// Grab values.  We don't know how many in advance so by default it calls push_back
file >> rolls;

std::cout << rolls << std::endl;

return 0;
}
```

Should produce output to the console depending on contents of file, e.g.,

```
{1,6,4,3,2,2,3,1,5,6}
```

7.20.2.6 template<typename T > file<file_type::input>& desalvo_standard_library::file< file_type::input >::read (T & *t*)

Loads in next value of type T from stream

Template Parameters

T	must have operator>>(istream&, T&) defined

Parameters

t	is an object reference

Returns

false if file is no longer in valid state

```
// Program to generate rectangular version of Hilbert matrix
#include "std_cin.h"
#include "std_cout.h"
#include "file.h"

namespace dsl = desalvo_standard_library;

// custom class to handle 2D data point, in order to demonstrate how to integrate custom code with existing
       library.
class Point2D {
// output operator: std::cout << pt << std::endl;
friend std::ostream& operator<<(std::ostream& out, const Point2D& pt) {
return out << "(" << pt.x << "," << pt.y << ")";
}

// input operator: std::cin >> pt;
friend std::istream& operator>>(std::istream& in, Point2D& pt) {
char unused;

// format is: (x,y)
// More generally: (char)x(char)y(char)
in >> unused;   // (
in >> pt.x;     //  x
in >> unused;   //   ,
in >> pt.y;     //    y
in >> unused;   //     )

return in;
}
public:
// Initialize value of point
Point2D(double input_x, double input_y) : x(input_x), y(input_y) { };

// Default constructor, calls more general constructor, new C++11.
Point2D() : Point2D(0,0) { };
private:
double x, y;
};

int main(int argc, const char * argv[]) {

// Local file path as std::string for easy concatenation
std::string prefix = "/Users/stephendesalvo/Documents/";
```

```
// For outputting to console
dsl::file<dsl::file_type::console> console;

// For reading in data from file
dsl::file<dsl::file_type::input> file(prefix + "data_collections.txt");

std::pair<int, int> my_pair;
std::vector<int> v;
std::set<double> v2;
std::set<int> v3;
std::multiset<int> v4;
std::list<Point2D> v5;
std::valarray<double> v6;
std::array<Point2D,4> v7;

console << "Get values in the same format as the default dsl output from file: data_collections.txt \n";
console << "Loading pair values: \n";         file.read(my_pair);
console << "Loading vector of ints: \n";       file.read(v);
console << "Loading set of doubles: \n";       file.read(v2);
console << "Loading set of ints: \n";          file.read(v3);
console << "Loading multiset of ints: \n";     file.read(v4);
console << "Loading list of Point2Ds: \n";     file.read(v5);
console << "Loading valarray of doubles: \n";  file.read(v6);
console << "Loading array of 4 Point2Ds: \n";  file.read(v7);

console << "pair stored as: ";      console.write(my_pair,std::endl);
console << "vector of ints: ";      console.write(v, std::endl);
console << "set of doubles: ";      console.write(v2,std::endl);
console << "set of ints: ";         console.write(v3,std::endl);
console << "multiset of ints: ";    console.write(v4,std::endl);
console << "list of Point2Ds: ";    console.write(v5,std::endl);
console << "valarray of doubles: "; console.write(v6,std::endl);
console << "array of Point2D: ";    console.write(v7,std::endl);

return 0;
}
```

The file data_collections.txt is

```
{1,2}
{5,4,2,1}
{1.1,1.1,2.2,5.4,6.5,3.14159}
{1,1,1,1,2,2,2,3,3,4,5}
{1,1,1,1,2,2,2,3,3,4,5}
{(0,0),(-1.1,1),(-2,2)}
{1.1,1.2,1.3,1.4,1.5}
{(0,0),(1,1),(2,2),(3,3)}
```

Should produce output

```
Get values in the same format as the default dsl output from file: data_collections.txt
Loading pair values:
Loading vector of ints:
Loading set of doubles:
Loading set of ints:
Loading multiset of ints:
Loading list of Point2Ds:
Loading valarray of doubles:
Loading array of 4 Point2Ds:
pair stored as: {1,2}
vector of ints: {5,4,2,1}
set of doubles: {1.1,2.2,3.14159,5.4,6.5}
set of ints: {1,2,3,4,5}
multiset of ints: {1,1,1,1,2,2,2,3,3,4,5}
list of Point2Ds: {(0,0),(-1.1,1),(-2,2)}
valarray of doubles: {1.1,1.2,1.3,1.4,1.5}
array of Point2D: {(0,0),(1,1),(2,2),(3,3)}
```

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

- DeSalvo Standard Library/desalvo/file.h

7.21 desalvo_standard_library::file< file_type::output > Class Template Reference

Public Member Functions

- file (std::string filename, std::fstream::openmode additional_modes=std::fstream::out, size_t output_-precision=10)

- file (file &&other)
- file & operator= (file &&other)
- ∼file ()
- template<typename T >
  file & **operator**<< (T &&t)
- template<typename T >
  file & **write** (T &&t)
- file & operator<< (manip1 fp)
- file & operator<< (manip2 fp)
- file & operator<< (manip3 fp)
- template<typename T >
  file & **operator**<< (const std::vector< T > &v)
- template<typename T >
  file< file_type::output > & operator<< (T &&t)
- template<typename T >
  file< file_type::output > & write (T &&t)

7.21.1 Constructor & Destructor Documentation

7.21.1.1 desalvo_standard_library::file< file_type::output >::file (std::string *filename*, std::fstream::openmode *additional_modes* = `std::fstream::out`, size_t *output_precision* = `10`)

Constructor via filename and various modes and options

Parameters

filename	is the full filepath of the file
additional_-modes	is a collection of options for formatting output
output_precision	is for outputting numerical floating point values

```
#include "desalvo/std_cout.h"
#include "desalvo/file.h"

#include <random>

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Local file path as std::string for easy concatenation
std::string prefix = "/Users/stephendesalvo/Documents/";

dsl::file< dsl::file_type::output > file(prefix + "data.txt");

file << 3;
file << '.';
file << 1<<4<<1<<5<<9<<2<<6<<5;

// too slow?
std::string more_digits = "35897932384626433832795028841971693993751058209";

file << more_digits;

return 0;
}
```

Doesn't produce any output to the console, but in the file data.txt we should see

```
3.1415926535897932384626433832795028841971693993751058209
```

7.21.1.2 desalvo_standard_library::file< file_type::output >::file (file< file_type::output > && *other*)

Move constructor

Parameters

other	is the expiring file stream

```
#include "desalvo/std_cout.h"
#include "desalvo/file.h"

#include <random>

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Local file path as std::string for easy concatenation
std::string prefix = "/Users/stephendesalvo/Documents/";

dsl::file< dsl::file_type::output > file(prefix + "data.txt");

file << 3;
file << '.';
file << 1<<4<<1<<5<<9<<2<<6<<5;

// too slow?
std::string more_digits = "358979323846264338327950288419716939937510582 09";

file << more_digits;

// Grab ownership of file, but ONLY via move constructor
auto file2( std::move(file) );

// Copy constructor is not defined, so line below will not compile
//auto file3(file2);

return 0;
}
```

Doesn't produce any output to the console, but in the file data.txt we should see

```
3.14159265358979323846264338327950288419716939937510582097494 4
```

7.21.1.3 desalvo_standard_library::file< file_type::output >::∼file ( )

Closes out the stream. +1 for RAII!

7.21.2 Member Function Documentation

7.21.2.1 file< file_type::output > & desalvo_standard_library::file< file_type::output >::operator<< (manip1 *fp*)

Output for manipulators, allows one to use std::endl as input

Parameters

fp	is an argument like std::endl

Returns

a reference to the file object for chaining

```
#include "desalvo/std_cout.h"
#include "desalvo/file.h"
#include <iomanip>

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Local file path as std::string for easy concatenation
std::string prefix = "/Users/stephendesalvo/Documents/";

dsl::file< dsl::file_type::output > file(prefix + "values.txt");

file << std::setprecision(20);
```

```
file << 3.14159265358979323846264338327 << std::endl;

file << std::setw(20) << std::left;

file << 3 << std::setw(20) << 12 << std::setw(10) << 4 << std::setw(5) << 9987;

return 0;
}
```

Doesn't produce any output to the console, but in the file data.txt we should see (something like)

```
3.141592653589793116
3                   12                  4         9987
```

7.21.2.2 file< file_type::output > & desalvo_standard_library::file< file_type::output >::operator<< (manip2 *fp*)

Output for manipulators

Parameters

fp	is an argument which manipulates the underlying file stream

Returns

a reference to the file object for chaining

```
#include "desalvo/std_cout.h"
#include "desalvo/file.h"
#include <iomanip>

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Local file path as std::string for easy concatenation
std::string prefix = "/Users/stephendesalvo/Documents/";

dsl::file< dsl::file_type::output > file(prefix + "values.txt");

file << std::setprecision(20);

file << 3.14159265358979323846264338327 << std::endl;

file << std::setw(20) << std::left;

file << 3 << std::setw(20) << 12 << std::setw(10) << 4 << std::setw(5) << 9987;

return 0;
}
```

Doesn't produce any output to the console, but in the file data.txt we should see (something like)

```
3.141592653589793116
3                   12                  4         9987
```

7.21.2.3 file< file_type::output > & desalvo_standard_library::file< file_type::output >::operator<< (manip3 *fp*)

Output for manipulators

Parameters

fp	is an argument which manipulates the underlying file stream

Returns

a reference to the file object for chaining

```
#include "desalvo/std_cout.h"
```

```
#include "desalvo/file.h"
#include <iomanip>

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Local file path as std::string for easy concatenation
std::string prefix = "/Users/stephendesalvo/Documents/";

dsl::file< dsl::file_type::output > file(prefix + "values.txt");

file << std::setprecision(20);

file << 3.14159265358979323846264338327 << std::endl;

file << std::setw(20) << std::left;

file << 3 << std::setw(20) << 12 << std::setw(10) << 4 << std::setw(5) << 9987;

return 0;
}
```

Doesn't produce any output to the console, but in the file data.txt we should see (something like)

```
3.141592653589793116
3                   12                  4         9987
```

7.21.2.4 template< typename T > file< file_type::output >& desalvo_standard_library::file< file_type::output >::operator<< (T && *t*)

Outputs argument to underlying file stream

Template Parameters

t	is the input for the file stream

Returns

a reference to the file object for chaining

```
#include "desalvo/std_cout.h"
#include "desalvo/file.h"

#include <random>

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Local file path as std::string for easy concatenation
std::string prefix = "/Users/stephendesalvo/Documents/";

dsl::file< dsl::file_type::output > file(prefix + "data.txt");

file << 3;
file << '.';
file << 1<<4<<1<<5<<9<<2<<6<<5;

// too slow?
std::string more_digits = "35897932384626433832795028841971693993751058209";

file << more_digits;

return 0;
}
```

Doesn't produce any output to the console, but in the file data.txt we should see

```
3.1415926535897932384626433832795028841971693993751058209
```

Example 2:

```
#include "desalvo/std_cout.h"
```

```
#include "desalvo/file.h"
#include <iomanip>

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Local file path as std::string for easy concatenation
std::string prefix = "/Users/stephendesalvo/Documents/";

dsl::file< dsl::file_type::output > file(prefix + "vector.txt");

std::vector<int> v {3,1,4,1,5,9,2,6,5};

file << v << std::endl;

return 0;
}
```

Doesn't produce any output to the console, but in the file vector.txt we should see

```
{3,1,4,1,5,9,2,6,5}
```

7.21.2.5 file< file_type::output > & desalvo_standard_library::file< file_type::output >::operator= (file< file_type::output > && *other*)

move assignment operator

Parameters

other	is the expiring file stream

Returns

reference to file for chaining

```
#include "desalvo/std_cout.h"
#include "desalvo/file.h"

#include <random>

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Local file path as std::string for easy concatenation
std::string prefix = "/Users/stephendesalvo/Documents/";

dsl::file< dsl::file_type::output > file(prefix + "data.txt");

file << 3;
file << '.';
file << 1<<4<<1<<5<<9<<2<<6<<5;

// too slow?
std::string more_digits = "35897932384626433832795028841971693993751058209";

file << more_digits;

dsl::file< dsl::file_type::output > file2(prefix + "data2.txt");

// Throw in some digits of e in another file
file2 << "2.718281828";

// close out digits of e file, take ownership of digits of pi file
file2 = std::move(file);

// Copy constructor is not defined, so line below will not compile
//file2 = file;

file2 << "7494459230";

return 0;
}
```

Doesn't produce any output to the console, but in the file data.txt we should see

```
3.1415926535897932384626433832795028841971693993751058209749445923O
```

and in the file data2.txt we should see

```
2.718281828
```

7.21.2.6 template<typename T > file<file_type::output>& desalvo_standard_library::file< file_type::output >::write (T && *t*)

Outputs argument to underlying file stream

Template Parameters

t	is the input for the file stream

Returns

a reference to the file object for chaining

```
#include "desalvo/std_cout.h"
#include "desalvo/file.h"

namespace dsl = desalvo_standard_library;

class Point2D {
friend std::ostream& operator<<(std::ostream& out, const Point2D& pt) {
return out << "(" << pt.x << "," << pt.y << ")";
}
public:
Point2D(double input_x, double input_y) : x(input_x), y(input_y) { };
private:
double x, y;
};

int main(int argc, const char * argv[]) {

// Local file path as std::string for easy concatenation
std::string prefix = "/Users/stephendesalvo/Documents/";

dsl::file< dsl::file_type::output > file(prefix + "data.txt");

file << 3;
file << '.';
file << 1<<4<<1<<5<<9<<2<<6<<5;

// too slow?
std::string more_digits = "358979323846264338327950288419716939937510582O9";

file << more_digits << std::endl;

double math_e = 2.718281828;

file.write(math_e);
file.write("\n");
file.write(Point2D(3,4));
file << std::endl;

return 0;
}
```

Doesn't produce any output to the console, but in the file data.txt we should see

```
3.14159265358979323846264338327950288419716939937510582O9
2.718281828
(3,4)
```

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

- DeSalvo Standard Library/desalvo/file.h

7.22 desalvo_standard_library::finite_sequence< storage, Derived, T, V > Class Template Reference

A CRTP class for working with finite sequences.

```
#include <sequence.h>
```

7.22.1 Detailed Description

template<dsl::store storage, typename Derived, typename T, typename V = std::vector<T>>class desalvo_standard_library::finite_sequence< storage, Derived, T, V >

A CRTP class for working with finite sequences.

The idea is to specify a sequence like permutations or Fibonacci numbers. There are two aspects to storage: 1. How to store a particular value in the sequence; 2. How to store the collection of values.

Each particular value is stored as type T, a template parameter.

If the storage is one at a time, then point 2 is moot since we don't store the entire sequence, and we only work with objects of type T. If the storage is random access, then the template parameter V determines how the collection is stored. Typically, V is an std::vector<T>, which is the default template parameter.

There are currently three options for storage: 1. Random access; 2. Bidirectional; 3. Forward only. Certain sequences like permutations can be easily incremented and decremented, whereas the partition numbers would need to be stored in a table in order to easily go backwards. On the other hand, permutations up to order 10 can be enumerated completely, and so a random access table is reasonable to pre-compute and quickly access random elements.

Template Parameters

storage	determines how the sequence is stored.
Derived	is assumed to be a class with member functions first_in_sequence(), last_in_-sequence(), and next_in_sequence(V& v) defined.
V	is assumed to have a nested iterator type which is a random access iterator.

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

- DeSalvo Standard Library/desalvo/sequence.h

7.23 desalvo_standard_library::finite_sequence< dsl::store::bidirectional, Derived, T, V > Class Template Reference

Classes

- class iterator

Public Member Functions

- iterator **begin** () const
- iterator **end** () const
- size_t **count** ()

Friends

- std::ostream & **operator**<< (std::ostream &out, const finite_sequence &seq)

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

- DeSalvo Standard Library/desalvo/sequence.h

7.24 desalvo_standard_library::finite_sequence< dsl::store::forward, Derived, T, V > Class Template Reference

Classes

- class iterator

Public Member Functions

- iterator **begin** () const
- iterator **end** () const
- size_t **count** ()

Friends

- std::ostream & **operator<<** (std::ostream &out, const finite_sequence &seq)

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

- DeSalvo Standard Library/desalvo/sequence.h

7.25 desalvo_standard_library::finite_sequence< dsl::store::random_access, Derived, T, V > Class Template Reference

Classes

- class iterator

Public Member Functions

- void **store_sequence** ()
- template<typename String = std::string>
  void **print** (std::ostream &out, String ending=std::string(""))
- size_t **size** () const
- T & **operator[]** (size_t rank)
- const T & **operator[]** (size_t rank) const
- iterator **begin** () const
- iterator **end** () const
- size_t **count** ()

Friends

- std::ostream & **operator<<** (std::ostream &out, const finite_sequence &seq)

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

- DeSalvo Standard Library/desalvo/sequence.h

7.26 desalvo_standard_library::finite_sequence_threadable< storage, Derived, T, V > Class Template Reference

A CRTP class for working with finite sequences.

```
#include <sequence.h>
```

7.26.1 Detailed Description

template<dsl::store storage, typename Derived, typename T, typename V = std::vector<T>>class desalvo_standard_library::finite_sequence_threadable< storage, Derived, T, V >

A CRTP class for working with finite sequences.

The idea is to specify a sequence like permutations or Fibonacci numbers. There are two aspects to storage: 1. How to store a particular value in the sequence; 2. How to store the collection of values.

Each particular value is stored as type T, a template parameter.

If the storage is one at a time, then point 2 is moot since we don't store the entire sequence, and we only work with objects of type T. If the storage is random access, then the template parameter V determines how the collection is stored. Typically, V is an std::vector<T>, which is the default template parameter.

There are currently three options for storage: 1. Random access; 2. Bidirectional; 3. Forward only. Certain sequences like permutations can be easily incremented and decremented, whereas the partition numbers would need to be stored in a table in order to easily go backwards. On the other hand, permutations up to order 10 can be enumerated completely, and so a random access table is reasonable to pre-compute and quickly access random elements.

Template Parameters

storage	determines how the sequence is stored.
Derived	is assumed to be a class with member functions first_in_sequence(), last_in_sequence(), and next_in_sequence(V& v) defined.
V	is assumed to have a nested iterator type which is a random access iterator.

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

- DeSalvo Standard Library/desalvo/sequence.h

7.27 desalvo_standard_library::finite_sequence_threadable< dsl::store::forward, Derived, T, V > Class Template Reference

Classes

- class iterator

Public Member Functions

- **finite_sequence_threadable** (size_t number_of_threads=std::thread::hardware_concurrency())
- iterator **begin** () const
- iterator **end** () const
- iterator **begin** (size_t i) const
- iterator **end** (size_t i) const
- void **count_by_threads** (iterator start, iterator stop)
- size_t **count_by_threads** ()

Friends

- std::ostream & **operator**<< (std::ostream &out, const finite_sequence_threadable &seq)

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

- DeSalvo Standard Library/desalvo/sequence.h

7.28 desalvo_standard_library::Fraction< T > Class Template Reference

Fraction class for storing int/int with arithmetic operators defined.

`#include <fraction.h>`

Public Member Functions

- Fraction ()
- template<typename F >
  Fraction (F t)
- template<typename F , typename G >
  Fraction (F t, G b)
- Fraction & operator+= (const Fraction &f)
- Fraction & operator-= (const Fraction &f)
- Fraction & operator*= (const Fraction &f)
- Fraction & operator/= (const Fraction &f)
- Fraction & operator++ ()
- Fraction operator++ (int)
- Fraction & operator-- ()
- Fraction operator-- (int)
- operator double ()

Friends

- bool operator== (const Fraction &lhs, const Fraction &rhs)
- bool operator< (const Fraction &lhs, const Fraction &rhs)
- std::ostream & **operator**<< (std::ostream &out, const Fraction &frac)

7.28.1 Detailed Description

template<typename T>class desalvo_standard_library::Fraction< T >

Fraction class for storing int/int with arithmetic operators defined.

This class attempts to mimic the usual mathematical notion of a fraction, allowing the programmer to write statements like a*b, a+=b, etc. for two Fraction objects a and b.

7.28.2 Constructor & Destructor Documentation

7.28.2.1 template<typename T > desalvo_standard_library::Fraction< T >::Fraction ( ) [inline]

Default Constructor for Fraction initializes to 0/1.

7.28.2.2 template<typename T > template<typename F > desalvo_standard_library::Fraction< T >::Fraction (F *t*) [inline]

Constructor initializer for int input

Parameters

t	is the numerator

7.28.2.3 template<typename T > template<typename F , typename G > desalvo_standard_library::Fraction< T >::Fraction (F *t*, G *b*) [inline]

Constructor for numerator/denominator inputs

Parameters

t	is the numerator
b	is the denominator

7.28.3 Member Function Documentation

7.28.3.1 template<typename T > desalvo_standard_library::Fraction< T >::operator double ( )

Conversion to double

7.28.3.2 template<typename T > Fraction< T > & desalvo_standard_library::Fraction< T >::operator*= (const Fraction< T > & *f*)

Operator for multiplying one fraction to another, overwriting the initial fraction

Parameters

f	is the value to multiply the object

Returns

a reference to the object being multiplied

7.28.3.3 template<typename T > Fraction< T > & desalvo_standard_library::Fraction< T >::operator++ ( )

The prefix increment operator for a fraction adds 1, or adds the denominator to the numerator

Returns

a reference to the Fraction object that called the operator

7.28.3.4 template<typename T > Fraction< T > desalvo_standard_library::Fraction< T >::operator++ (int)

The postfix increment operator for a fraction adds 1, or adds the denominator to the numerator

Parameters

unused	parameter to indicate postfix

Returns

a new Fraction object with the pre-incremented value

7.28.3.5 template<typename T > Fraction< T > & desalvo_standard_library::Fraction< T >::operator+= (const Fraction< T > & *f*)

Operator for adding one fraction to another, overwriting the initial fraction

Parameters

f	is the value to add to the object

Returns

a reference to the object being added to

7.28.3.6 template<typename T > Fraction< T > & desalvo_standard_library::Fraction< T >::operator-- ( )

The prefix decrement operator for a fraction subtracts 1, or subtracts the denominator to the numerator

Returns

a reference to the Fraction object that called the operator

7.28.3.7 template<typename T > Fraction< T > desalvo_standard_library::Fraction< T >::operator-- (int)

The postfix decrement operator for a fraction subtracts 1, or subtracts the denominator from the numerator

Parameters

unused	parameter to indicate postfix

Returns

a new Fraction object with the pre-decremented value

7.28.3.8 template<typename T > Fraction< T > & desalvo_standard_library::Fraction< T >::operator-= (const Fraction< T > & *f*)

Operator for subtracting one fraction to another, overwriting the initial fraction

Parameters

f	is the value to subtract from the object

Returns

a reference to the object being subtracted from

7.28.3.9 template< typename T > Fraction< T > & desalvo_standard_library::Fraction< T >::operator/= (const Fraction< T > & *f*)

Operator for dividing by one fraction to another, overwriting the initial fraction

Parameters

f	is the value to divide by the object

Returns

a reference to the object being divided by

7.28.4 Friends And Related Function Documentation

7.28.4.1 template< typename T> bool operator< (const Fraction< T > & *lhs,* const Fraction< T > & *rhs*) [friend]

Cross multiplies and compares the resulting integers. WARNING! This is not the best method, as there may be overflow. It would be more numerically stable to reduce the fractions to lowest terms and compare the numerators and denominators directly.

Parameters

lhs	is the left object
rhs	is the right object

Returns

true if the cross products correspond to strict inequality, false otherwise

7.28.4.2 template< typename T> bool operator== (const Fraction< T > & *lhs,* const Fraction< T > & *rhs*) [friend]

Cross multiplies and compares the resulting integers. WARNING! This is not the best method, as there may be overflow. It would be more numerically stable to reduce the fractions to lowest terms and compare the numerators and denominators directly.

Parameters

lhs	is the left object
rhs	is the right object

Returns

true if the cross products are the same, false otherwise

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

- DeSalvo Standard Library/desalvo/fraction.h

7.29 desalvo_standard_library::latin_square< ValueType, WorkingPrecision >::generator< Parameters > Class Template Reference

Public Member Functions

- **generator** (Parameters initial_p)
- template<typename URNG = std::mt19937_64>
  latin_square::object< Parameters > **operator()** (URNG &gen=generator_64)

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

- DeSalvo Standard Library/desalvo/latin_square.h

7.30 desalvo_standard_library::set_partition< KeyType, ValueType, WorkingPrecision >::generator< Parameters > Class Template Reference

Public Member Functions

- **generator** (ValueType initial_n)
- template<typename URNG >
  object< Parameters > **operator()** (SimulationMethod method=SimulationMethod::PDCDSH, URNG &generator=dsl::generator_64)

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

- DeSalvo Standard Library/desalvo/set_partition.h

7.31 desalvo_standard_library::integer_partition< UnsignedInteger > Class Template Reference

Public Member Functions

- integer_partition (UnsignedInteger n)
- integer_partition ()
- void clear ()
- void **print** (std::ostream &out=std::cout)
- UnsignedInteger **weight** ()
- integer_partition (UnsignedInteger n)
- integer_partition ()
- void clear ()
- void **print** (std::ostream &out=std::cout)
- UnsignedInteger **weight** ()

Friends

- class **integer_partition_generator< UnsignedInteger >**

7.31.1 Constructor & Destructor Documentation

7.31.1.1 template<typename UnsignedInteger > desalvo_standard_library::integer_partition< UnsignedInteger >::integer_partition (UnsignedInteger *n*) [inline]

Initialize partition to 1 part of size n

Parameters

n	is the size of the partition

7.31.1.2 template<typename UnsignedInteger > desalvo_standard_library::integer_partition< UnsignedInteger >::integer_partition ( ) [inline]

By default everything is empty

7.31.1.3 template<typename UnsignedInteger > desalvo_standard_library::integer_partition< UnsignedInteger >::integer_partition (UnsignedInteger *n*) [inline]

Initialize partition to 1 part of size n

Parameters

n	is the size of the partition

7.31.1.4 template<typename UnsignedInteger > desalvo_standard_library::integer_partition< UnsignedInteger >::integer_partition ( ) [inline]

By default everything is empty

7.31.2 Member Function Documentation

7.31.2.1 template<typename UnsignedInteger > void desalvo_standard_library::integer_partition< UnsignedInteger >::clear ( ) [inline]

Clear the partition to be empty

7.31.2.2 template<typename UnsignedInteger > void desalvo_standard_library::integer_partition< UnsignedInteger >::clear ( ) [inline]

Clear the partition to be empty

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

- DeSalvo Standard Library/desalvo/integer_partition.h
- DeSalvo Standard Library/desalvo/Recycle/integer_partition.h

7.32 desalvo_standard_library::integer_partition_generator< UnsignedInteger, Floating > Class Template Reference

Public Member Functions

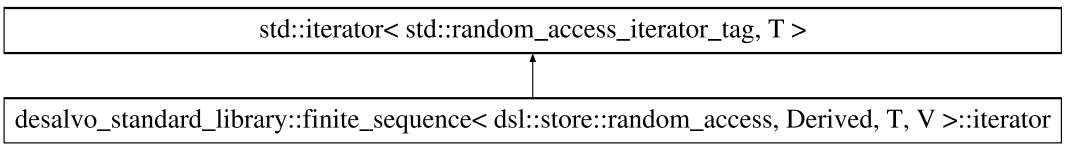
- **integer_partition_generator** (UnsignedInteger input_n)
- template<typename URNG >
  integer_partition
  < UnsignedInteger > **Boltzmann_sampler** (URNG &&generator)
- template<typename URNG >
  integer_partition
  < UnsignedInteger > **exact_Boltzmann_sampler** (URNG &&generator)
- dsl::table< double > **recursive_method_table** (UnsignedInteger n, UnsignedInteger k)
- **integer_partition_generator** (UnsignedInteger input_n)
- template<typename URNG >
  integer_partition
  < UnsignedInteger > **Boltzmann_sampler** (URNG &&generator)
- template<typename URNG >
  integer_partition
  < UnsignedInteger > **exact_Boltzmann_sampler** (URNG &&generator)
- dsl::table< double > **recursive_method_table** (UnsignedInteger n, UnsignedInteger k)

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

- DeSalvo Standard Library/desalvo/integer_partition.h
- DeSalvo Standard Library/desalvo/Recycle/integer_partition.h

7.33 desalvo_standard_library::finite_sequence< dsl::store::random_access, Derived, T, V >::iterator Class Reference

Inheritance diagram for desalvo_standard_library::finite_sequence< dsl::store::random_access, Derived, T, V >::iterator:



Public Member Functions

- **iterator** (const finite_sequence &seq)
- **iterator** (const finite_sequence &seq, size_t index)
- **iterator** (const iterator &it)
- void **swap** (iterator &other)
- iterator & **operator=** (iterator it)
- const T & **operator[]** (size_t index) const
- const T ∗ **operator->** () const
- const T & **operator∗** () const
- iterator & **operator+=** (int increment)
- iterator & **operator-=** (int increment)
- iterator & **operator++** ()
- iterator **operator++** (int unused)
- iterator & **operator--** ()
- iterator **operator--** (int unused)

Friends

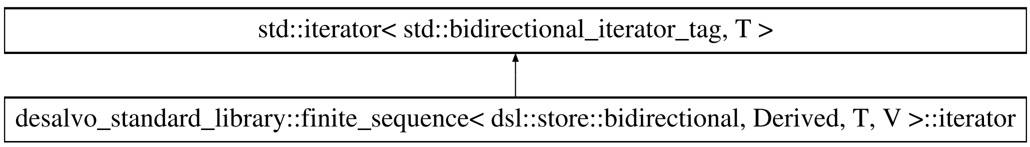
- bool **operator==** (const iterator &lhs, const iterator &rhs)
- bool **operator!=** (const iterator &lhs, const iterator &rhs)
- bool **operator<** (const iterator &lhs, const iterator &rhs)
- bool **operator<=** (const iterator &lhs, const iterator &rhs)
- bool **operator>** (const iterator &lhs, const iterator &rhs)
- bool **operator>=** (const iterator &lhs, const iterator &rhs)
- std::iterator
  < std::random_access_iterator_tag,
  T >::difference_type **operator-** (const iterator &lhs, const iterator &rhs)
- iterator **operator+** (iterator lhs, int increment)
- iterator **operator+** (int increment, iterator lhs)
- iterator **operator-** (iterator lhs, int increment)
- iterator **operator-** (int increment, iterator lhs)

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

- DeSalvo Standard Library/desalvo/sequence.h

7.34 desalvo_standard_library::finite_sequence< dsl::store::bidirectional, Derived, T, V >::iterator Class Reference

Inheritance diagram for desalvo_standard_library::finite_sequence< dsl::store::bidirectional, Derived, T, V >:-:iterator:



Public Member Functions

- **iterator** (const finite_sequence &seq)
- **iterator** (const finite_sequence &seq, T ∗heap_element)
- **iterator** (const iterator &it)
- void **swap** (iterator &other)
- iterator & **operator=** (iterator it)
- const T ∗ **operator->** () const
- const T & **operator∗** () const
- iterator & **operator++** ()
- iterator **operator++** (int unused)
- iterator & **operator--** ()
- iterator **operator--** (int unused)

Friends

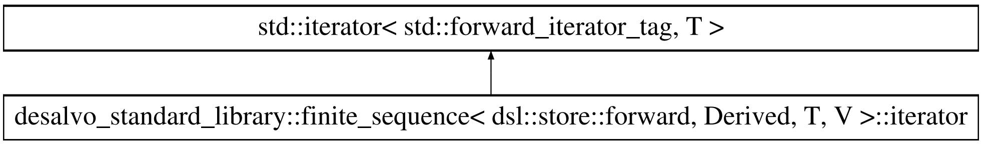
- bool **operator==** (const iterator &lhs, const iterator &rhs)
- bool **operator!=** (const iterator &lhs, const iterator &rhs)

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

- DeSalvo Standard Library/desalvo/sequence.h

7.35 desalvo_standard_library::finite_sequence< dsl::store::forward, Derived, T, V >::iterator Class Reference

Inheritance diagram for desalvo_standard_library::finite_sequence< dsl::store::forward, Derived, T, V >::iterator:



Public Member Functions

- **iterator** (const finite_sequence &seq)
- **iterator** (const finite_sequence &seq, T ∗heap_element)
- **iterator** (const iterator &it)
- void **swap** (iterator &other)
- iterator & **operator=** (iterator it)
- const T ∗ **operator->** () const
- const T & **operator∗** () const
- iterator & **operator++** ()
- iterator **operator++** (int unused)

Friends

- bool **operator==** (const iterator &lhs, const iterator &rhs)
- bool **operator!=** (const iterator &lhs, const iterator &rhs)

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

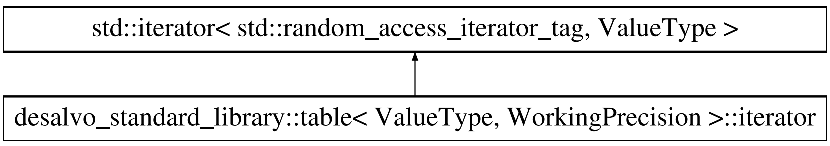
- DeSalvo Standard Library/desalvo/sequence.h

7.36 desalvo_standard_library::table< ValueType, WorkingPrecision >::iterator Class Reference

random access iterator for a table, treating it like a 1D array

```
#include <table.h>
```

Inheritance diagram for desalvo_standard_library::table< ValueType, WorkingPrecision >::iterator:



Public Member Functions

- iterator (table ∗initial_mat=nullptr, int initial_row=0, int initial_col=0)
- iterator (const iterator &other)

- void swap (iterator &other)
- iterator & operator= (iterator to_copy)
- iterator & operator++ ()
- iterator operator++ (int)
- iterator & operator+= (int col)
- iterator operator+ (int col)
- iterator & operator-- ()
- iterator operator-- (int)
- iterator & operator-= (int col)
- iterator operator- (int col)
- int operator- (const iterator &p2) const
- ValueType & operator* ()
- ValueType & operator-> ()
- ValueType & operator[] (int n)

Friends

- bool operator== (const table::iterator &lhs, const table::iterator &rhs)
- bool operator< (const table::iterator &lhs, const table::iterator &rhs)

7.36.1 Detailed Description

template<typename ValueType = double, typename WorkingPrecision = long double>class desalvo_standard_library::table< ValueType, WorkingPrecision >::iterator

random access iterator for a table, treating it like a 1D array

Use this if the tabular structure of the table is immaterial, and you would like to operate on entries of the table as if they were one contiguous array.

```
#include <iostream>
#include <vector>
#include "std_cout.h"
#include "desalvo/table.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Create 10x10 table, default initialized values (i.e., 0 for int)
dsl::table<int> t(10,10);

// Very simple linear congruential engine
int A = 16807;
int C = 127;
int value = 1;

// initialize values using custom linear congruential engine
for(auto& i : t)
i = (value = ( (A*value)%C));

std::cout << "t: \n" << t << std::endl << std::endl;

// Sort the entire set of values
std::sort(t.begin(), t.end());

std::cout << "Sort all elements, and print t again:\n";
std::cout << t << std::endl << std::endl;

// Check for duplicates
auto start = t.cbegin();
auto stop = t.cend();

// check is initialized one before start, which now points to second element
auto checker = start++;

unsigned int number_of_duplicates = 0;

// loop through all elements, essentially treating table as 1D array
```

```
while(start != stop) {

// check for equailty, increment
if(*checker++ == *start++)
++number_of_duplicates;
}

std::cout << "The number of duplicate elements is: " << number_of_duplicates << std::endl << std::endl;

// I get 0 duplicates, which is not very random!
// This is an example of the birthday problem.  At each new element created, there is a chance it will
       match an existing element.  Assuming perfect randomness, the probability that there are no duplicates in 100
       values of 127 possible values is given by
// (1)(1-1/127)(1-2/127)...(1-99/127)
// Let's do that calculation quickly.
double x = 1.;

for(size_t i=1;i<100;++i)
x *= 1.-i/127.;

std::cout << "The probability of having 0 elements if they were in fact generated perfectly randomly is: "
      << x << ".\n\n";

std::cout << "Do you need further anecdotal evidence to suggest that linear congruential generators are not
       great randomizers?" << std::endl;

return 0;
}
```

Should produce output

```
t:
{{43,71,5,88,101,25,59,124,125,41},
{112,117,78,52,77,9,6,4,45,30},
{20,98,23,100,109,115,119,37,67,87},
{58,81,54,36,24,16,53,120,80,11},
{92,19,55,79,95,21,14,94,105,70},
{89,17,96,64,85,99,66,44,114,76},
{93,62,126,84,56,122,39,26,102,68},
{3,2,86,15,10,49,75,50,118,121},
{123,82,97,107,29,104,27,18,12,8},
{90,60,40,69,46,73,91,103,111,74}}

Sort all elements, and print t again:
{{2,3,4,5,6,8,9,10,11,12},
{14,15,16,17,18,19,20,21,23,24},
{25,26,27,29,30,36,37,39,40,41},
{43,44,45,46,49,50,52,53,54,55},
{56,58,59,60,62,64,66,67,68,69},
{70,71,73,74,75,76,77,78,79,80},
{81,82,84,85,86,87,88,89,90,91},
{92,93,94,95,96,97,98,99,100,101},
{102,103,104,105,107,109,111,112,114,115},
{117,118,119,120,121,122,123,124,125,126}}

The number of duplicate elements is: 0

The probability of having 0 elements if they were in fact generated perfectly randomly is: 1.15238e-25.

Do you need further anecdotal evidence to suggest that linear congruential generators are not great
      randomizers?
```

7.36.2 Constructor & Destructor Documentation

7.36.2.1 template<typename ValueType = double, typename WorkingPrecision = long double> desalvo_standard_library::table< ValueType, WorkingPrecision >::iterator::iterator (table * *initial_mat* = nullptr, int *initial_row* = 0, int *initial_col* = 0) [inline]

Constructors with various default parameters.

If 2 parameters are provided, then the default column is 0.

If 1 prameter is provided, then the default row and column are 0.

If no parameters are provided, then the default row and column are 0 and the default table pointer is set to nullptr.

Parameters

initial_mat	is a table for which to refer to
initial_row	is the initial row entry to refer to
initial_col	is the initial column entry to refer to

7.36.2.2 template<typename ValueType = double, typename WorkingPrecision = long double> desalvo_standard_library::table< ValueType, WorkingPrecision >::iterator::iterator (const iterator & *other*) [inline]

Default Copy constructor, no memory is being managed so this one can also be generated by the compiler.

Parameters

other	is the other iterator from which to copy values.

7.36.3 Member Function Documentation

7.36.3.1 template<typename ValueType = double, typename WorkingPrecision = long double> ValueType& desalvo_standard_library::table< ValueType, WorkingPrecision >::iterator::operator* ( ) [inline]

Accesses the entry directly relative to mat, so even if mat assumes new memory or location it still points to relatively the same location

Returns

reference to (row,column)-th entry in mat

7.36.3.2 template<typename ValueType = double, typename WorkingPrecision = long double> iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::iterator::operator+ (int *col*) [inline]

Increment operator for iterator, so that it works like a pointer.

Parameters

col	is an increment through columns, can be negative

Returns

a new iterator referring to the element indicated by the increment.

7.36.3.3 template<typename ValueType = double, typename WorkingPrecision = long double> iterator& desalvo_standard_library::table< ValueType, WorkingPrecision >::iterator::operator++ ( ) [inline]

Standard prefix ++ operator

Returns

reference to self after the increment.

7.36.3.4 template<typename ValueType = double, typename WorkingPrecision = long double> iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::iterator::operator++ (int) [inline]

Postfix ++ operator

7.36.3.5 template< typename ValueType = double, typename WorkingPrecision = long double > iterator& desalvo_standard_library::table< ValueType, WorkingPrecision >::iterator::operator+= (int *col*) [inline]

Increment equals operator for iterator, so that it works like a pointer.

Parameters

col	is an increment through columns, can be negative

Returns

reference to the iterator for chaining.

7.36.3.6 template< typename ValueType = double, typename WorkingPrecision = long double > iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::iterator::operator- (int *col*) [inline]

Decrement operator for iterator, so that it works like a pointer.

Parameters

col	is a decrement through columns, can be negative

Returns

a new iterator referring to the element indicated by the decrement.

7.36.3.7 template< typename ValueType = double, typename WorkingPrecision = long double > int desalvo_standard_library::table< ValueType, WorkingPrecision >::iterator::operator- (const iterator & *p2*) const [inline]

Take the difference between two iterators of the same type, ∗this-p2

Parameters

p2	is the rhs of ∗this-p2

Returns

the number of elements that must be transversed in order to get from ∗this to p2; can be negative.

7.36.3.8 template< typename ValueType = double, typename WorkingPrecision = long double > iterator& desalvo_standard_library::table< ValueType, WorkingPrecision >::iterator::operator-- ( ) [inline]

Standard prefix – operator

Returns

reference to self after the increment.

7.36.3.9 template< typename ValueType = double, typename WorkingPrecision = long double > iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::iterator::operator-- (int) [inline]

Postfix – operator

7.36.3.10 template<typename ValueType = double, typename WorkingPrecision = long double> iterator& desalvo_standard_library::table< ValueType, WorkingPrecision >::iterator::operator-= (int *col*) [inline]

Decrement equals operator for iterator, so that it works like a pointer.

Parameters

col	is a decrement through columns, can be negative

Returns

reference to the iterator for chaining.

7.36.3.11 template<typename ValueType = double, typename WorkingPrecision = long double> ValueType& desalvo_standard_library::table< ValueType, WorkingPrecision >::iterator::operator-> ( ) [inline]

Accesses the entry directly relative to mat, so even if mat assumes new memory or location it still points to relatively the same location

Returns

reference to (row,column)-th entry in mat

7.36.3.12 template<typename ValueType = double, typename WorkingPrecision = long double> iterator& desalvo_standard_library::table< ValueType, WorkingPrecision >::iterator::operator= (iterator *to_copy*) [inline]

Assignment operator, follows the usual copy and swap idiom

Parameters

to_copy	is an iterator from whcih to copy from.

Returns

a reference to the iterator for chaining.

7.36.3.13 template<typename ValueType = double, typename WorkingPrecision = long double> ValueType& desalvo_standard_library::table< ValueType, WorkingPrecision >::iterator::operator[] (int *n*) [inline]

Accesses the entry n past the current point

Returns

reference to (row,column+n)-th entry in mat

7.36.3.14 template<typename ValueType = double, typename WorkingPrecision = long double> void desalvo_standard_library::table< ValueType, WorkingPrecision >::iterator::swap (iterator & *other*) [inline]

Swaps out the values of two iterators, even iterators referring to different tables.

Parameters

other	is the table from which to swap.

7.36.4 Friends And Related Function Documentation

7.36.4.1 template<typename ValueType = double, typename WorkingPrecision = long double> bool operator< (const table::iterator & *lhs*, const table::iterator & *rhs*) [friend]

Tests for iterators in two equivalent positions

Parameters

t	is the other iterator

Returns

true if iterators are equivalent

7.36.4.2 template<typename ValueType = double, typename WorkingPrecision = long double> bool operator== (const table::iterator & *lhs*, const table::iterator & *rhs*) [friend]

Tests for iterators in two equivalent positions

Parameters

t	is the other iterator

Returns

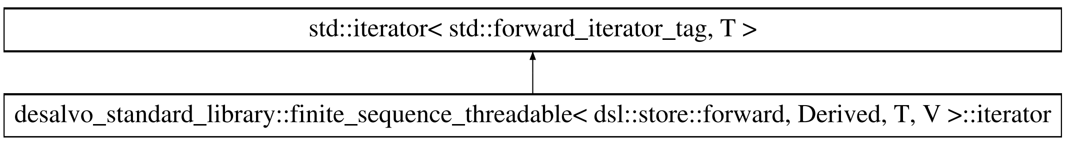
true if iterators are equivalent

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

- DeSalvo Standard Library/desalvo/table.h

7.37 desalvo_standard_library::finite_sequence_threadable< dsl::store::forward, Derived, T, V >::iterator Class Reference

Inheritance diagram for desalvo_standard_library::finite_sequence_threadable< dsl::store::forward, Derived, T, V >::iterator:



Public Member Functions

- **iterator** (const finite_sequence_threadable &seq)
- **iterator** (const finite_sequence_threadable &seq, size_t in_thread)
- **iterator** (const finite_sequence_threadable &seq, T ∗heap_element)

- **iterator** (const iterator &it)
- **iterator** (iterator &&it)
- void **swap** (iterator &other)
- iterator & **operator=** (iterator it)
- const T ∗ **operator->** () const
- const T & **operator∗** () const
- iterator & **operator++** ()
- iterator **operator++** (int unused)

Friends

- bool **operator==** (const iterator &lhs, const iterator &rhs)
- bool **operator!=** (const iterator &lhs, const iterator &rhs)

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

- DeSalvo Standard Library/desalvo/sequence.h

7.38 desalvo_standard_library::latin_square< ValueType, WorkingPrecision >::iterator_bidirectional< Parameters > Class Template Reference

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

- DeSalvo Standard Library/desalvo/latin_square.h

7.39 desalvo_standard_library::latin_square< ValueType, WorkingPrecision >::iterator_forward< Parameters > Class Template Reference

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

- DeSalvo Standard Library/desalvo/latin_square.h

7.40 desalvo_standard_library::latin_square< ValueType, WorkingPrecision >::iterator_random_access< Parameters > Class Template Reference

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

- DeSalvo Standard Library/desalvo/latin_square.h

7.41 desalvo_standard_library::latin_square< ValueType, WorkingPrecision > Class Template Reference

Classes

- class generator
- class iterator_bidirectional
- class iterator_forward
- class iterator_random_access
- class object

Public Member Functions

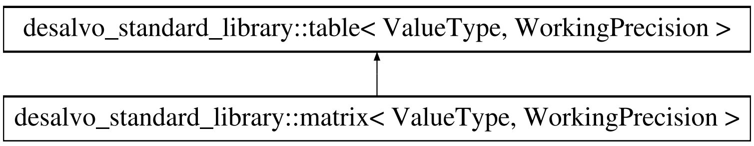
- **latin_square** (size_t input_n=0)
- template<typename Parameters = std::vector<int>>
  object< Parameters > **first** (ValueType N)
- template<typename Parameters = std::vector<int>>
  object< Parameters > **last** ()
- template<typename Parameters = std::vector<int>, typename URNG = std::mt19937_64>
  object< Parameters > **random** (ValueType n, URNG &gen=generator_64)

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

- DeSalvo Standard Library/desalvo/latin_square.h

7.42 desalvo_standard_library::matrix< ValueType, WorkingPrecision > Class Template Reference

Inheritance diagram for desalvo_standard_library::matrix< ValueType, WorkingPrecision >:



Public Member Functions

- matrix ()
- matrix (size_t number_of_rows, size_t number_of_columns=1, const ValueType &val=ValueType())
- **matrix** (const matrix &initial_matrix)
- virtual ~matrix ()
- matrix & operator*= (const ValueType &value)
- matrix & operator/= (const ValueType &value)
- matrix & operator+= (const matrix &rhs)
- matrix & operator-= (const matrix &rhs)
- matrix **operator-** () const
- matrix **operator+** () const
- void **transpose** ()
- WorkingPrecision **power_iteration** (size_t max_iters=10000)
- double second_largest_eigenvalue_of_stochastic_square_matrix (size_t max_iters=10000)

Friends

- std::ostream & **operator<<** (std::ostream &out, const matrix &t)
- matrix operator* (const ValueType &value, matrix m)
- matrix operator/ (const ValueType &value, matrix m)
- matrix operator* (const matrix &lhs, const matrix &rhs)
- matrix operator+ (matrix lhs, const matrix &rhs)
- matrix operator- (matrix lhs, const matrix &rhs)
- bool **operator==** (const matrix &lhs, const matrix &rhs)

7.42.1 Constructor & Destructor Documentation

7.42.1.1 template<typename ValueType = double, typename WorkingPrecision = long double> desalvo_standard_library::matrix< ValueType, WorkingPrecision >::matrix ( ) [inline]

Initializes the "empty" matrix.

7.42.1.2 template<typename ValueType = double, typename WorkingPrecision = long double> desalvo_standard_library::matrix< ValueType, WorkingPrecision >::matrix (size_t *number_of_rows*, size_t *number_of_columns* = 1, const ValueType & *val* = ValueType()) [inline]

Initialize entries to value

Parameters

val	is the initial value for all entries.

7.42.1.3 template<typename ValueType = double, typename WorkingPrecision = long double> virtual desalvo_standard_library::matrix< ValueType, WorkingPrecision >::~matrix ( ) [inline], [virtual]

virtual destructors deletes entry memory.

7.42.2 Member Function Documentation

7.42.2.1 template<typename ValueType = double, typename WorkingPrecision = long double> matrix& desalvo_standard_library::matrix< ValueType, WorkingPrecision >::operator*= (const ValueType & *value*) [inline]

Multiplication by a scalar, A = A*c

Parameters

value	is the scalar c

Returns

reference to newly updated matrix B which is still (M x N)

7.42.2.2 template<typename ValueType = double, typename WorkingPrecision = long double> matrix& desalvo_standard_library::matrix< ValueType, WorkingPrecision >::operator+= (const matrix< ValueType, WorkingPrecision > & *rhs*) [inline]

(R x M) + (R x M) addition, C = A+B

Parameters

lefty	is the (M x M) matrix A

Returns

reference to newly updated matrix B which is still (M x N)

7.42.2.3 template<typename ValueType = double, typename WorkingPrecision = long double> matrix& desalvo_standard_library::matrix< ValueType, WorkingPrecision >::operator-= (const matrix< ValueType, WorkingPrecision > & *rhs*) [inline]

(R x M) + (R x M) addition, C = A+B

Parameters

lefty	is the (M x M) matrix A

Returns

reference to newly updated matrix B which is still (M x N)

7.42.2.4 template<typename ValueType = double, typename WorkingPrecision = long double> matrix& desalvo_standard_library::matrix< ValueType, WorkingPrecision >::operator/= (const ValueType & *value*) [inline]

Multiplication by a scalar, A = A∗c

Parameters

value	is the scalar c

Returns

reference to newly updated matrix B which is still (M x N)

7.42.2.5 template<typename ValueType , typename WorkingPrecision > double desalvo_standard_library::matrix< ValueType, WorkingPrecision >::second_largest_eigenvalue_of_stochastic_square_matrix (size_t *max_iters* = 10000)

Use Wielandt Deflation, Algorithm 9.4 in Burden-Faires 7th Edition to find second largest eigenvalue by constructing a smaller matrix and then applying the power method on it.

Template Parameters

ValueType	is the data type stored in the matrix
WorkingPrecision	is the type used for numerical calculations

Parameters

max_iters	is the maximum number of iterations for the power method on the smaller matrix

7.42.3 Friends And Related Function Documentation

7.42.3.1 template<typename ValueType = double, typename WorkingPrecision = long double> matrix operator∗ (const ValueType & *value*, matrix< ValueType, WorkingPrecision > *m*) [friend]

Multiplication by a scalar, A = A∗c

Parameters

value	is the scalar c

Returns

reference to newly updated matrix B which is still (M x N)

7.42.3.2 template<typename ValueType = double, typename WorkingPrecision = long double> matrix operator∗ (const matrix< ValueType, WorkingPrecision > & *lhs*, const matrix< ValueType, WorkingPrecision > & *rhs*) [friend]

(R x M) ∗ (M x N) multiplication, B = A∗B

Parameters

lefty	is the (M x M) matrix A

Returns

reference to newly updated matrix B which is still (M x N)

7.42.3.3 template<typename ValueType = double, typename WorkingPrecision = long double> matrix operator+ (matrix< ValueType, WorkingPrecision > *lhs*, const matrix< ValueType, WorkingPrecision > & *rhs*) [friend]

(R x M) + (R x M) addition, C = A+B

Parameters

lefty	is the (M x M) matrix A

Returns

reference to newly updated matrix B which is still (M x N)

7.42.3.4 template<typename ValueType = double, typename WorkingPrecision = long double> matrix operator- (matrix< ValueType, WorkingPrecision > *lhs*, const matrix< ValueType, WorkingPrecision > & *rhs*) [friend]

(R x M) + (R x M) addition, C = A+B

Parameters

lefty	is the (M x M) matrix A

Returns

reference to newly updated matrix B which is still (M x N)

7.42.3.5 template<typename ValueType = double, typename WorkingPrecision = long double> matrix operator/ (const ValueType & *value*, matrix< ValueType, WorkingPrecision > *m*) [friend]

Multiplication by a scalar, A = A∗c

Parameters

value	is the scalar c

Returns

reference to newly updated matrix B which is still (M x N)

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

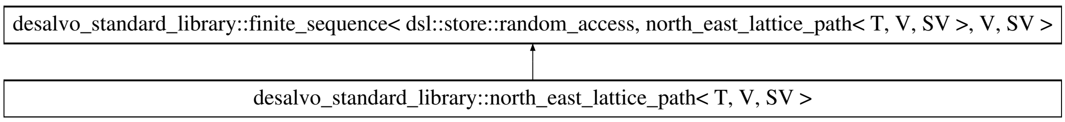
- DeSalvo Standard Library/desalvo/matrix.h

7.43 desalvo_standard_library::north_east_lattice_path< T, V, SV > Class Template Reference

all walks from (0,0) to (n,k) using up and right moves

```
#include <combinatorics.h>
```

Inheritance diagram for desalvo_standard_library::north_east_lattice_path< T, V, SV >:



Public Member Functions

- **north_east_lattice_path** (size_t input_n, size_t input_k)
- V **first_in_sequence** () const
- bool **next_in_sequence** (V &v) const

7.43.1 Detailed Description

template<typename T = bool, typename V = std::vector<T>, typename SV = std::vector<V>>class desalvo_standard_library::north_east_lattice_path< T, V, SV >

all walks from (0,0) to (n,k) using up and right moves

I wanted to enumerate all of the paths.

Example:

```
// Code to check the n choose k different sets, make sure each are occurring equally likely
#include "desalvo/dsl_usings.h" // See documentation for list of keywords included in
       this file.

int main(int argc, const char * argv[]) {

// from (0,0) to (n,k)
size_t n = 6;
size_t k = 3;
size_t m = 100000;

std::multiset< std::vector<bool> > s;

// Insert each generated set into s
for(size_t i=0;i<m;++i)
s.insert(dsl::set_n_choose_k(n,k));

// Generate all possible paths from (0,0) to (n,k)
dsl::north_east_lattice_path<bool> paths(n,k);

// Print out each path along with the number of times it occurs in s
for(auto& x : paths)
std::cout << x << ": " << s.count(x) << std::endl;

return 0;
}
```

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

- DeSalvo Standard Library/desalvo/combinatorics.h

7.44 desalvo_standard_library::NotDivisibleBy Class Reference

Creates function objects which check for divisibility.

`#include <numerical.h>`

Public Member Functions

- NotDivisibleBy (unsigned long in)
- bool operator() (unsigned long x)

7.44.1 Detailed Description

Creates function objects which check for divisibility.

```
#include "desalvo/numerical.h" // See documentation for list of keywords included in
       this file.
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Form matrix [[1,2,3],[4,5,6],[7,8,9]] using a row-major 1D vector
std::vector<int> v {1,2,3,4,5,6,7,8,9,10};
std::vector<int> v2(10);

auto it = std::copy_if( std::begin(v), std::end(v), std::begin(v2),
      dsl::NotDivisibleBy(3) );

// optional erase, makes output cleaner
v2.erase(it, std::end(v2));


dsl::print(v,"\n");
dsl::print(v2,"\n");


return 0;
}
```

Should produce output

```
{1,2,3,4,5,6,7,8,9,10}
{1,2,4,5,7,8,10}
```

7.44.2 Constructor & Destructor Documentation

7.44.2.1 desalvo_standard_library::NotDivisibleBy::NotDivisibleBy (unsigned long *in*) [inline]

Construct a function object with a given divisibility condition

Parameters

in	is the divisibility value

7.44.3 Member Function Documentation

7.44.3.1 bool desalvo_standard_library::NotDivisibleBy::operator() (unsigned long *x*) [inline]

Checks if input is divisible by n

Parameters

x	is the input to check for divisibility

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

- DeSalvo Standard Library/desalvo/numerical.h

7.45 desalvo_standard_library::numeric_data< T, Container > Class Template Reference

stores collections of numeric data, calculates statistics

`#include <statistics.h>`

Public Member Functions

- template<typename F >
  void **add_point** (F &&data_point)
- template<typename F , typename String = std::string>
  void **print_points** (F &&out=std::cout, String &&sep=String(","), String &&open_bracket=String("{"), String &&close_bracket=String("}")) const
- void **write_to_file** (std::string filename) const
- template<typename RealType = T>
  RealType **mean** () const

Public Attributes

- Container **points**

Friends

- std::ostream & **operator<<** (std::ostream &out, const numeric_data &dp)

7.45.1 Detailed Description

template<typename T, typename Container = std::vector<T>>class desalvo_standard_library::numeric_data< T, Container >

stores collections of numeric data, calculates statistics

This class is designed to quickly obtain and process numerical data and provide various simple statistics.

Example 1:

```
#include <iostream>
#include "desalvo/dsl_usings.h"

int main(int argc, const char * argv[]) {

    dsl::numeric_data<double> points;
    points.add_point(5);
    points.add_point(12);
    points.add_point(13);
    points.add_point(2);
    points.add_point(-123.3);
```

```
    points.print_points(std::cout, ",", "[","]"); std::cout << std::endl;

    std::cout << points << std::endl;
    std::cout << points.mean() << std::endl;

    return 0;
}
```

Output:

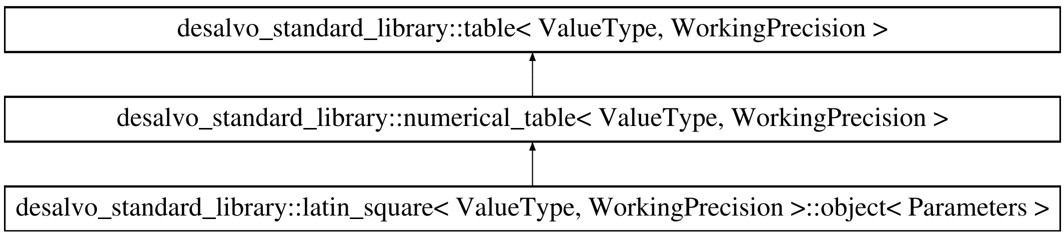
```
[5,12,13,2,-123.3]
{5,12,13,2,-123.3}
-18.26
```

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

- DeSalvo Standard Library/desalvo/statistics.h

7.46 desalvo_standard_library::numerical_table< ValueType, WorkingPrecision > Class Template Reference

Inheritance diagram for desalvo_standard_library::numerical_table< ValueType, WorkingPrecision >:



Public Member Functions

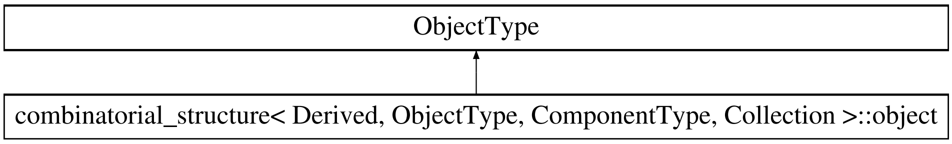
- **numerical_table** (size_t m=0, size_t n=0)
- numerical_table & **operator+=** (const numerical_table &rhs)
- numerical_table & **operator-=** (const numerical_table &rhs)
- template<typename T >
  numerical_table & **operator+=** (const T &value)
- template<typename T >
  numerical_table & **operator-=** (const T &value)
- numerical_table **operator-** ()
- template<typename ValueTypeLocal = long double, typename WorkingPrecisionLocal = long double>
  numerical_table **operator+** ()

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

- DeSalvo Standard Library/desalvo/numerical_table.h

7.47 combinatorial_structure< Derived, ObjectType, ComponentType, Collection >::object Class Reference

Inheritance diagram for combinatorial_structure< Derived, ObjectType, ComponentType, Collection >::object:

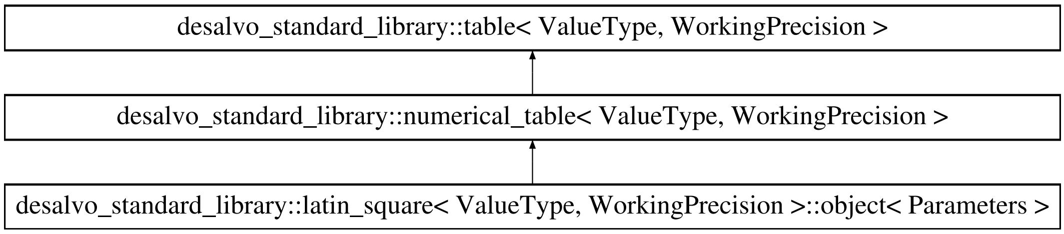


The documentation for this class was generated from the following file:

- DeSalvo Standard Library/desalvo/combinatorial_structure.h

7.48 desalvo_standard_library::latin_square< ValueType, WorkingPrecision >::object< Parameters > Class Template Reference

Inheritance diagram for desalvo_standard_library::latin_square< ValueType, WorkingPrecision >::object< Parameters >:



Public Member Functions

- **object** (size_t N)

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

- DeSalvo Standard Library/desalvo/latin_square.h

7.49 desalvo_standard_library::set_partition< KeyType, ValueType, WorkingPrecision >::object< Parameters > Class Template Reference

Public Member Functions

- **object** (ValueType initial_size)
- std::map< KeyType, ValueType > ::iterator **begin** ()
- std::map< KeyType, ValueType > ::iterator **end** ()
- ValueType **weight** ()
- ValueType **number_of_components** ()

Friends

- class **generator**< **Parameters** >

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

- DeSalvo Standard Library/desalvo/set_partition.h

7.50 permutation Class Reference

container for restrictions of the form {none, fixed_point_free, by_pairs, by_function}

`#include <documentation.h>`

7.50.1 Detailed Description

container for restrictions of the form {none, fixed_point_free, by_pairs, by_function}

Storage can be either random access, bidirectional, forward.

Random access is useful for small n, since at present it loops through all n! permutations much be stored in memory. There are more efficient iteratations through restricted permutations, but they have not been implemented in this version.

Example 1:

```
const int n = 4;

fixed_point_free_permutation<size_t> v(n);
std::cout << v.size() << std::endl;

// Print out the 5th fixed-point free permutation of 4
std::cout << v[4] << std::endl;

// Add up the first two elements of each permutation.
for(auto& x : v) std::cout << x[0]+x[1] << ",";
std::cout << std::endl;

// Supplies a random access iterator, so can do pointer arithmetic
//auto start = std::begin(v);
//auto stop = std::end(v);
//auto it = start + 3;

// We can access each permutation using generic algorithms.
std::for_each(std::begin(v), std::end(v), [](const std::vector<size_t>& v) {std::cout << v << std::endl;});

// Requires a random access iterator, can search for particular elements.
auto flag = std::binary_search(std::begin(v), std::end(v), std::vector<size_t>({4,3,2,1}));
std::cout << flag << std::endl;
```

Example 2:

```
fixed_point_free_permutation_one<size_t> v(4);

// Add up the first two elements of each permutation.
//for(auto& x : v) std::cout << x[0]+x[1] << ",";
//std::cout << std::endl;

// Supplies a random access iterator, so can do pointer arithmetic
//auto start = std::begin(v);
//auto stop = std::end(v);
//auto it = start + 3;

auto t = std::end(v);

--t;

auto s = std::begin(v);

--s;
++s;

std::cout << *s << std::endl;

// We can access each permutation using generic algorithms.
std::for_each(std::begin(v), std::end(v), [](const std::vector<size_t>& v) {std::cout << v << std::endl;});
```

```
// Requires a random access iterator, can search for particular elements.
auto flag = std::binary_search(std::begin(v), std::end(v), std::vector<size_t>({4,3,2,1}));
std::cout << flag << std::endl;

std::cout << v.count() << std::endl;
```

Example 3:

```
#include "desalvo/dsl_usings.h" // See documentation for list of keywords included in
       this file.



int main(int argc, const char * argv[]) {

dsl::permutation<dsl::store::forward, dsl::restrictions::fixed_point_free>
       p(6);

dsl::permutation<dsl::store::forward, dsl::restrictions::none>
       p2(4);

dsl::permutation<dsl::store::random_access, dsl::restrictions::by_pairs>
       p3(5, {{1,2},{2,4}});
dsl::permutation<dsl::store::forward, dsl::restrictions::by_pairs>
       p4(10, {{1,2},{2,4}});
p4.insert({{1,1},{2,2},{3,3},{4,4},{5,5},{2,1}});


//auto start = std::begin(p);
//auto stop = std::end(p);

for(auto& x : p)
std::cout << x << std::endl;

for(auto& x : p2)
std::cout << x << std::endl;


for(auto& x : p3)
std::cout << x << std::endl;

std::cout << "\n\n";

std::cout << p3.size() << std::endl;

p3.insert({{1,1},{2,2},{3,3},{4,4},{5,5},{2,1}});

for(auto& x : p3)
std::cout << x << std::endl;

std::cout << p3.size() << std::endl;

p3.resize(6);

for(auto& x : p3)
std::cout << x << std::endl;

std::cout << p3.size() << std::endl;

auto start = std::begin(p4);

for(size_t i=0;i<100;++i)
std::cout << *start++ << std::endl;


//auto start = std::begin(p3);
//auto stop = std::end(p3);

//std::cout << *start << std::endl;


return 0;
}
```

Example 4:

```
#include "desalvo/dsl_usings.h" // See documentation for list of keywords included in
       this file.

// avoiding 132
bool avoids_132(std::vector<size_t>& v) {
```

```
size_t n = v.size();

// Look at all triplets of indices to see if pattern is violated.
// O(n^3) algorithm.
for(size_t i=0;i<n-2;++i)
for(size_t j=i+1;j<n-1;++j)
for(size_t k=j+1;k<n;++k)
if( (v[i] < v[j] && v[i] < v[k] && v[j] > v[k]) )
return true;

return false;

}

// avoiding 132 consecutively
bool avoids_132_consecutively(std::vector<size_t>& v) {

size_t n = v.size();

// Look at all consecutive triplets of indices to see if pattern is violated.
// O(n) algorithm.
for(size_t i=0;i<n-2;++i)
if( (v[i] < v[i+1] && v[i] < v[i+2] && v[i+1] > v[i+2]) )
return true;

return false;

}

int main(int argc, const char * argv[]) {

dsl::permutation<dsl::store::random_access, dsl::restrictions::by_function>
       p(6, avoids_132);
dsl::permutation<dsl::store::random_access, dsl::restrictions::by_function>
       p_consecutive(6, avoids_132_consecutively);


for(auto& x : p)
std::cout << x << std::endl;

std::cout << "\n\n";

for(auto& x : p_consecutive)
std::cout << x << std::endl;



return 0;
}
```

Example 5:

```
#include "desalvo/dsl_usings.h" // See documentation for list of keywords included in
       this file.

// avoiding 132
bool avoids_132(std::vector<size_t>& v) {

size_t n = v.size();

// Look at all triplets of indices to see if pattern is violated.
// O(n^3) algorithm.
for(size_t i=0;i<n-2;++i)
for(size_t j=i+1;j<n-1;++j)
for(size_t k=j+1;k<n;++k)
if( (v[i] < v[j] && v[i] < v[k] && v[j] > v[k]) )
return true;

return false;

}

int main(int argc, const char * argv[]) {

dsl::permutation<dsl::store::bidirectional, dsl::restrictions::by_function>
       p(11, avoids_132);

// Should return the 11-th Catalan number: 58786
std::cout << p.count() << std::endl;


return 0;
}
```

Example 6: This code was used to update the values a(13)-a(14) in the sequence OEIS A165546.

```
#include "desalvo/dsl_usings.h" // See documentation for list of keywords included in
       this file.

//function to test whether a permutation avoids both 3412 AND 2413.
bool avoids_3412_and_2413(const std::vector<size_t>& v) {

size_t n = v.size();

// Look at all quadruplets of indices to see if pattern is violated.
// O(n^4) algorithm.
for(size_t h=0;h<n-3;++h)
for(size_t i=h+1;i<n-2;++i)
for(size_t j=i+1;j<n-1;++j)
for(size_t k=j+1;k<n;++k)
if( (v[j] < v[k] && v[k] < v[h] && v[h] < v[i]) ||
(v[j] < v[h] && v[h] < v[k] && v[k] < v[i]) )
return true;

return false;

}

int main(int argc, const char * argv[]) {

// Start your engines ...
dsl::time t;

// Create a collection of permutations of {1,2,...,13}, and the ability to iterate forward from a given
       valid permutation.
size_t i = 13;
dsl::permutation<dsl::store::forward, dsl::restrictions::by_function>
       p(i, avoids_3412_and_2413);

// Calculate the number of elements in the set, threaded since we defined begin(i) and end(i) for
       i=1,2,...,n
std::cout << i << ": " << p.count_by_threads() << ", ";
std::cout << "in " << t.toc() << " seconds \n";
std::cout.flush();

// Repeat for permutations of {1,2,...,14}
i = 14;
dsl::permutation<dsl::store::forward, dsl::restrictions::by_function>
       p2(i, avoids_3412_and_2413);

std::cout << i << ": " << p2.count_by_threads() << ", ";
std::cout << "in " << t.toc() << " seconds \n";
std::cout.flush();

return 0;
}
```

Example 7:

```
// Code to check the n choose k different sets, make sure each are occurring equally likely
#include "desalvo/dsl_usings.h" // See documentation for list of keywords included in
       this file.


//function to test whether a permutation avoids both 3412 AND 2413.
bool every_321_extends_to_3241(const std::vector<size_t>& v) {

size_t n = v.size();

bool flag = true;

if(v[0] == 3 && v[1] == 5 && v[2] == 2 && v[3] == 4 && v[4] == 1) {
//std::cout << v << std::endl;
}

// Look at all quadruplets of indices to see if pattern is violated.
// O(n^4) algorithm.
for(size_t i=0;i<n-2;++i)
for(size_t j=i+1;j<n-1;++j)
for(size_t k=j+1;k<n;++k) {
//std::cout << v << std::endl;
if( (v[i] > v[j] && v[j] > v[k] ) ) {

flag = false; // reset flag

for(size_t s = j+1;s<k;++s)
if(v[s]>v[i]) {
flag = true;
break;
}
```

```
if(!flag) return true;
}
}

return false;

}

int main(int argc, const char * argv[]) {

// Start your engines ...
dsl::time t;

// The condition that every occurrence of 321 extends to an occurrence of 3241 implies the set of
       permutations is in one-to-one correspondence to the Bell numbers.
size_t i = 5;
dsl::permutation<dsl::store::forward, dsl::restrictions::by_function>
       p(i, every_321_extends_to_3241);

for(auto& x : p)
std::cout << x << ",";
std::cout << std::endl;

// Calculate the number of elements in the set, threaded since we defined begin(i) and end(i) for
       i=1,2,...,n
std::cout << i << ": " << p.count_by_threads() << ", ";
std::cout << "in " << t.toc() << " seconds \n";
std::cout.flush();


return 0;
}
```

Example 8:

```
// Attempt to speed up pattern-avoiding permutation calculation using a local cache.  Didn't really work
       like I hoped it would, even though optimized to not need locks or mutexes.

#include "desalvo/dsl_usings.h" // See documentation for list of keywords included in
       this file.

// Store local cache of values which caused violations.  Restart from scratch every time the cache exceeds
       say 100 entries.

#include <map>

// max number of threads
size_t num_threads = 20;

// max number of stored violations
size_t max_caches = 100;
std::vector< std::vector< std::vector<size_t> > > cache(num_threads, std::vector<std::vector<size_t>>(
      max_caches,{0,0,0,0}));
std::vector< size_t > current_size(num_threads);

size_t id = 1;
std::map< std::thread::id, size_t > mapping;


//function to test whether a permutation avoids both 3412 AND 2413.
bool avoids_3412_and_2413(const std::vector<size_t>& v) {

    size_t n = v.size();

    // Establish an integer-correspondence with each thread, since thread ids are not convertible to int.
    // would ideally need to optimize this away, since not necessary to keep doing this every function
       call.
    // Only accounted for 4.2% of time.
    if(mapping[std::this_thread::get_id()] == 0)
    {
        mapping[std::this_thread::get_id()] = id++;
        //std::cout << "id = " << id << "mapping = " << mapping[std::this_thread::get_id()] << std::endl;

    }

    // Get current thread index
    size_t thread = mapping[std::this_thread::get_id()];


    // Go through elements in the cache first, since recently found violations are likely to continue
       occurring in future calls.
    for(size_t ii=0,nn=current_size[thread];ii<nn;++ii) {

        size_t h = cache[thread][ii][0];
        size_t i = cache[thread][ii][1];
```

```
        size_t j = cache[thread][ii][2];
        size_t k = cache[thread][ii][3];

        if( (v[j] < v[k] && v[k] < v[h] && v[h] < v[i]) ||
            (v[j] < v[h] && v[h] < v[k] && v[k] < v[i])     )
            return true;
    }

    // Look at all quadruplets of indices to see if pattern is violated.
    // O(n^4) algorithm.
    for(size_t h=0;h<n-3;++h)
        for(size_t i=h+1;i<n-2;++i) {
            // early abort condition
            while(h < n-3 && i < n-2 && v[h] > v[i]) {
                ++i;
                if(i == n-2) {
                    ++h;
                    i=h+1;
                }
            }
            for(size_t j=i+1;j<n-1;++j)
                for(size_t k=j+1;k<n;++k)

                    // check condition
                    if( (v[j] < v[k] && v[k] < v[h] && v[h] < v[i]) ||
                        (v[j] < v[h] && v[h] < v[k] && v[k] < v[i]) ) {

                        // If we are running this code, we have a violation!

                        // clear cache if it gets too big.
                        if(current_size[thread] >= max_caches) {
                            //std::cout << "cache reset" << std::endl;

                            // Rather than calling .clear, we reset the max relevant value
                            //cache.clear();
                            current_size[thread] = 0;
                        }

                        // update cached value for future reference.
                        cache[thread][current_size[thread]++] = std::vector<size_t>({h,i,j,k});
                        return true;
                    }
        }

    // no violations occurred
    return false;

}

int main(int argc, const char * argv[]) {


    // Start your engines ...
    dsl::time t;

    // Create a collection of permutations of {1,2,...,13}, and the ability to iterate forward from a given
       valid permutation.
    size_t i = 11;
    dsl::permutation<dsl::store::forward, dsl::restrictions::by_function>
       p(i, avoids_3412_and_2413);

    //for(auto& x : p)
    //    std::cout << x << std::endl;

    // Calculate the number of elements in the set, threaded since we defined begin(i) and end(i) for
       i=1,2,...,n
    std::cout << i << ": " << p.count_by_threads() << ", ";
    //std::cout << i << ": " << p.count() << ", ";
    std::cout << "in " << t.toc() << " seconds \n";
    std::cout.flush();


    return 0;
}
```

Example 9:

```
// Generates iid samples of random Mallows(q) permutations for q = qmin, ..., qmax and checks how many
       avoid 321 consecutively.  The variable n is the size of the permutation and m is the number of samples.


#include "desalvo/dsl_usings.h" // See documentation for list of keywords included in
       this file.

// avoiding 321 consecutively
```

```
bool avoids_321_consecutively(const std::vector<size_t>& v) {

    size_t n = v.size();

    // Look at all consecutive triplets of indices to see if pattern is violated.
    // O(n) algorithm.
    for(size_t i=0;i<n-3;++i)
        if( (v[i] > v[i+1] && v[i+1] > v[i+2]) )
            return false;

    return true;

}

int main(int argc, const char * argv[]) {

    // Create mesh grid
    double qmin = 0.01;
    double qmax = 1.;
    size_t mesh_size = 20;
    std::vector<double> vals(mesh_size+1);

    // n is the size of the permutation, m is the number of iterations.
    size_t n = 30;
    size_t m = 100;

    // keeps track of which value of q we are using in the vector
    size_t index = 0;

    // q = qmin, qmin+delta, qmin+2delta, ..., qmax
    for(long double q = qmin; q <= qmax; q += (qmax-qmin)/(mesh_size-1)) {

        double avoids_321 = 0.; // count number that avoid 321

        for(size_t i=0;i<m;++i)

            // generate permutation using Mallows(q) distribution, test for whether it avoids 321
       consecutively.
            if(avoids_321_consecutively(dsl::random_permutation_mallows(n, q
      , dsl::generator_64)))
                avoids_321 = avoids_321 + 1.;

        // Keep track of which ones avoid, store the average^(1/n)
        if(avoids_321)
            vals[index++] = std::pow(avoids_321/m,1./n);
        else
            vals[index++] = 1.;
    }

    dsl::print(vals,"[", ",", "]");

    //    std::cout << std::pow(avoids_321,-1./n) << std::endl;

    return 0;
}
```

Example 10:

```
#include "desalvo/dsl_usings.h" // See documentation for list of keywords included in
       this file.

// NOT WORKING YET!  In process of being generalized from m=3 to general m
//  consecutively
bool avoids_all_m(size_t m, size_t n) {

    // generate all permutations of size m
    auto patterns = dsl::permutations<>( dsl::range(m) );

    size_t m_size = patterns.size();

    // set up vector of bools to check which patterns occurred over all permutations of n
    std::vector<bool> occurred(m_size);

    dsl::permutation<forward> permutations(n);

    // for each permutation of n ...
    for(auto& x : permutations) {

        for(size_t i=0;i<n-2;++i)
            for(size_t j=i+1;j<n-1;++j)
                for(size_t k=j+1;k<n;++k)
                    for(size_t s=0,r=patterns.size(); s<r;++s)

                        if( dsl::permutation_reduction({x[i],x[j],x[k]}) ==
      patterns[s]) {
```

```
                            occurred[s] = true;
                            s = r; // stop this one

                            // check if all conditions met
                            if(std::all_of(occurred.begin(), occurred.end(), true))
                                return true;
                        }
    }

    return false;

}

// Look at all consecutive triplets of indices to see if pattern is violated.
// O(n) algorithm.

return true;

}

int main() {

    size_t nmax = 12;

    std::vector<std::vector<size_t>> superpatterns(nmax, std::vector<size_t>(nmax));

    for(size_t i=2;i<=nmax;++i) {

        // All permutations of {1,2,..,i}
        dsl::permutation<forward> permutations(i);


        for(size_t j=2;j<=i;++i) {
            // all patterns of lengths 2, ..., i
            dsl::permutation<forward> patterns(j);



        }
    }

    //std::cout << v << std::endl;

    std::cout << reduction({1, 30, 500, 2, 4, 8, 11, 123, 50}) << std::endl;



    return 0;
}
```

Example 11:

```
// Generates iid samples of random Mallows(q) permutations for q = qmin, ..., qmax and estimates the
       probability of avoiding a given pattern of size 3. The variable n is the size of the permutation and m is the
       number of samples.

#include "desalvo/dsl_usings.h" // See documentation for list of keywords included in
       this file.
#include <map>


// avoiding 123
template<typename T>
bool avoids_123_consecutively(const std::vector<T>& v) {

    size_t n = v.size();

    // Look at all triplets of indices to see if pattern is violated.
    // O(n) algorithm.
    for(size_t i=0;i<n-2;++i)
        if( v[i] < v[i+1] && v[i+1] < v[i+2] ){
            //std::cout << std::vector<size_t>({i,j,k}) << std::endl;
            return false;
        }
    //return true;

    return true;

}

// avoiding 132
template<typename T>
bool avoids_132_consecutively(const std::vector<T>& v) {

    size_t n = v.size();
```

```
    // Look at all triplets of indices to see if pattern is violated.
    // O(n) algorithm.
    for(size_t i=0;i<n-2;++i)
        if( (v[i] < v[i+1] && v[i] < v[i+2] && v[i+1] > v[i+2]) ){
            //std::cout << std::vector<size_t>({i,j,k}) << std::endl;
            return false;
        }
    //return true;

    return true;

}



// avoiding 213
template<typename T>
bool avoids_213_consecutively(const std::vector<T>& v) {

    size_t n = v.size();

    // Look at all triplets of indices to see if pattern is violated.
    // O(n) algorithm.
    for(size_t i=0;i<n-2;++i)
        if( (v[i] > v[i+1] && v[i] < v[i+2] && v[i+1] < v[i+2]) ){
            //std::cout << std::vector<size_t>({i,j,k}) << std::endl;
            return false;
        }
    //return true;

    return true;

}



// avoiding 132
template<typename T>
bool avoids_231_consecutively(const std::vector<T>& v) {

    size_t n = v.size();

    // Look at all triplets of indices to see if pattern is violated.
    // O(n) algorithm.
    for(size_t i=0;i<n-2;++i)
        if( (v[i] < v[i+1] && v[i] > v[i+2] && v[i+1] > v[i+2]) ){
            //std::cout << std::vector<size_t>({i,j,k}) << std::endl;
            return false;
        }
    //return true;

    return true;

}



// avoiding 132
template<typename T>
bool avoids_312_consecutively(const std::vector<T>& v) {

    size_t n = v.size();

    // Look at all triplets of indices to see if pattern is violated.
    // O(n) algorithm.
    for(size_t i=0;i<n-2;++i)
        if( (v[i] > v[i+1] && v[i] > v[i+2] && v[i+1] < v[i+2]) ){
            //std::cout << std::vector<size_t>({i,j,k}) << std::endl;
            return false;
        }
    //return true;

    return true;

}


// avoiding 132
template<typename T>
bool avoids_321_consecutively(const std::vector<T>& v) {

    size_t n = v.size();

    // Look at all triplets of indices to see if pattern is violated.
    // O(n) algorithm.
    for(size_t i=0;i<n-2;++i)
```

```
        if( (v[i] > v[i+1] && v[i+1] > v[i+2]) ){
            //std::cout << std::vector<size_t>({i,j,k}) << std::endl;
            return false;
        }
    //return true;

    return true;

}


int main(int argc, const char * argv[]) {



    // Create mesh grid
    double qmin = .01;
    double qmax = 2.;
    size_t mesh_size = 51;
    std::vector<std::vector<double>> vals(6, std::vector<double>(mesh_size-1));
    std::vector<double> qs(mesh_size-1);

    // n is the size of the permutation, m is the number of iterations.
    size_t n = 20;
    size_t m = 100000;

    // keeps track of which value of q we are using in the vector
    size_t index = 0;

    std::map<int, size_t> pattern_to_counts;

    // q = qmin, qmin+delta, qmin+2delta, ..., qmax
    for(long double q = qmin; q < qmax; q += (qmax-qmin)/(mesh_size-1)) {

        qs[index] = q;

        pattern_to_counts[123] = 0;
        pattern_to_counts[132] = 0;
        pattern_to_counts[213] = 0;
        pattern_to_counts[231] = 0;
        pattern_to_counts[312] = 0;
        pattern_to_counts[321] = 0;

        //double avoids_321 = 0.; // count number that avoid 321

        for(size_t i=0;i<m;++i) {

            // generate random permutation in Mallows form
            auto v = dsl::random_permutation_mallows(n, q);
            //auto v = dsl::random_permutation(n);

            // generate permutation using Mallows(q) distribution, test for whether it avoids 321
      consecutively.
            if(avoids_123_consecutively(v)) ++pattern_to_counts[123];
            if(avoids_132_consecutively(v)) ++pattern_to_counts[132];
            if(avoids_213_consecutively(v)) ++pattern_to_counts[213];
            if(avoids_231_consecutively(v)) ++pattern_to_counts[231];
            if(avoids_312_consecutively(v)) ++pattern_to_counts[312];
            if(avoids_321_consecutively(v)) ++pattern_to_counts[321];
        }

        // Keep track of which ones avoid, store the average^(1/n)
        if(pattern_to_counts[123] > 0) vals[0][index] = std::pow((double)pattern_to_counts[123]/m,1./n);
        else vals[0][index] = 1.;

        // Keep track of which ones avoid, store the average^(1/n)
        //if(pattern_to_counts[123] > 0) vals[0][index] = (double)pattern_to_counts[123]/m;
        //else vals[0][index] = 1.;

        if(pattern_to_counts[132] > 0) vals[1][index] = std::pow((double)pattern_to_counts[132]/m,1./n);
        else vals[1][index] = 1.;

        if(pattern_to_counts[213] > 0) vals[2][index] = std::pow((double)pattern_to_counts[213]/m,1./n);
        else vals[2][index] = 1.;

        if(pattern_to_counts[231] > 0) vals[3][index] = std::pow((double)pattern_to_counts[231]/m,1./n);
        else vals[3][index] = 1.;

        if(pattern_to_counts[312] > 0) vals[4][index] = std::pow((double)pattern_to_counts[312]/m,1./n);
        else vals[4][index] = 1.;

        if(pattern_to_counts[321] > 0) vals[5][index] = std::pow((double)pattern_to_counts[321]/m,1./n);
        else vals[5][index] = 1.;

        ++index;

    }
```

```
    for(size_t i=0;i<6;++i) {

        std::cout << "ListLinePlot[Transpose[";
        std::cout << "{";
        dsl::print(qs,"{", ",", "}");
        std::cout << ",";
        dsl::print(vals[i],"{", ",", "}");
        std::cout << "}";
        std::cout << "],PlotRange->{" <<
        (double)(floor((*std::min_element(vals[i].begin(), vals[i].end()))*10.))/10.;
        std::cout << ",1},PlotLabel->\"";

        switch(i) {
            case 0 : std::cout << "123"; break;
            case 1 : std::cout << "132"; break;
            case 2 : std::cout << "213"; break;
            case 3 : std::cout << "231"; break;
            case 4 : std::cout << "312"; break;
            case 5 : std::cout << "321"; break;
        }

        std::cout << "\"]";

        std::cout << std::endl;

    }

    return 0;
}
```

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

- DeSalvo Standard Library/desalvo/documentation.h

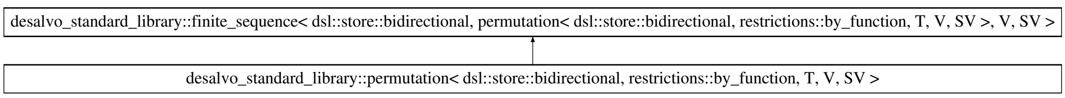
7.51 desalvo_standard_library::permutation< seq, type, T, V, SV > Class Template Reference

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

- DeSalvo Standard Library/desalvo/permutation.h

7.52 desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::by_function, T, V, SV > Class Template Reference

Inheritance diagram for desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::by_function, T, V, SV >:



Public Member Functions

- permutation (size_t input_n)
- permutation (size_t input_n, std::function< bool(V &)> restriction_function)
- void replace_restriction_function (std::function< bool(V &)> new_restriction_function)
- void resize (size_t n)
- V first_in_sequence () const
- V last_in_sequence () const
- bool next_in_sequence (V &v) const
- bool previous_in_sequence (V &v) const
- operator bool ()

7.52.1 Constructor & Destructor Documentation

7.52.1.1 template<typename T , typename V , typename SV > desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::by_function, T, V, SV >::permutation (size_t *input_n*)

Initializes permutation to have size n, computes the first element in the sequence, and stores the entire sequence.

Parameters

input_n	is the initial size of the permutation.

7.52.1.2 template<typename T , typename V , typename SV > desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::by_function, T, V, SV >::permutation (size_t *input_n*, std::function< bool(V &)> *initialize_restrictions*)

Initializes permutation to have size n, initializes the restrictions, and computes the sequence.

Parameters

input_n	is the initial size of the permutation
initialize_-restrictions	is the initial set of restrictions

7.52.2 Member Function Documentation

7.52.2.1 template<typename T , typename V , typename SV > V desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::by_function, T, V, SV >::first_in_sequence ( ) const

Compute the first instance of a permutation with given restrictions in lexicographic ordering

Returns

the first permutation in lexicographic ordering with the given restrictions

7.52.2.2 template<typename T , typename V , typename SV > V desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::by_function, T, V, SV >::last_in_sequence ( ) const

Compute the last instance of a permutation with given restrictions in lexicographic ordering

Returns

the last permutation in lexicographic ordering with the given restrictions

7.52.2.3 template<typename T , typename V , typename SV > bool desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::by_function, T, V, SV >::next_in_sequence (V & *v*) const

Given a current state, updates the input to the next state, returns false if next state restarts the sequence.

Parameters

v	is the input state, which is updated to the next state

Returns

whether or not the sequence restarted

7.52.2.4 template<typename T , typename V , typename SV > desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::by_function, T, V, SV >::operator bool ( )

conversion to bool, returns true unless the restrictions are too tough and no such set of permutations exist.

Returns

true if there are no elements, false otherwise

7.52.2.5 template<typename T , typename V , typename SV > bool desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::by_function, T, V, SV >::previous_in_sequence (V & *v*) const

Given a current state, updates the input to the previous state, returns false if previous state restarts the sequence.

Parameters

v	is the input state, which is updated to the previous state

Returns

whether or not the sequence restarted

7.52.2.6 template<typename T , typename V , typename SV > void desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::by_function, T, V, SV >::replace_restriction_function (std::function< bool(V &)> *new_restriction_function*)

replaces restriction function with the one input, recalculates the set of objects

Parameters

new_restriction-_function	is the new function to test against violations

7.52.2.7 template<typename T , typename V , typename SV > void desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::by_function, T, V, SV >::resize (size_t *input_n*)

Resizes the permutation and then reinitializes with the set of permutations

Parameters

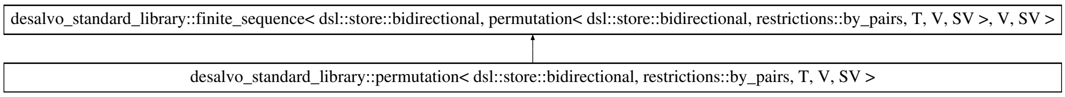
input_n	is the new size of the permutation

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

- DeSalvo Standard Library/desalvo/permutation.h

7.53 desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::by_pairs, T, V, SV > Class Template Reference

Inheritance diagram for desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::by_pairs, T, V, SV >:



Public Member Functions

- permutation (size_t input_n)
- permutation (size_t input_n, const std::set< std::pair< size_t, size_t > > &initialize_restrictions)
- permutation (size_t input_n, std::initializer_list< std::pair< size_t, size_t > > initial_list)
- void resize (size_t n)
- V first_in_sequence () const
- V last_in_sequence () const
- bool next_in_sequence (V &v) const
- bool previous_in_sequence (V &v) const
- void insert (const std::pair< size_t, size_t > &res)
- void insert (std::pair< size_t, size_t > &&res)
- void insert (std::initializer_list< std::pair< size_t, size_t > > res)
- template<typename InputIterator >
  void insert (InputIterator start, InputIterator stop)
- void **clear** ()
- operator bool ()

7.53.1 Constructor & Destructor Documentation

7.53.1.1 template<typename T , typename V , typename SV > **desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::by_pairs, T, V, SV >::permutation** (size_t *input_n*)

Initializes permutation to have size n, computes the first element in the sequence, and stores the entire sequence.

Parameters

input_n	is the initial size of the permutation.

7.53.1.2 template<typename T , typename V , typename SV > **desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::by_pairs, T, V, SV >::permutation** (size_t *input_n,* const std::set< std::pair< size_t, size_t > > & *initialize_restrictions*)

Initializes permutation to have size n, initializes the restrictions, and computes the sequence.

Parameters

input_n	is the initial size of the permutation
initialize_restrictions	is the initial set of restrictions

7.53.1.3 template< typename T , typename V , typename SV > desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::by_pairs, T, V, SV >::permutation (size_t *input_n*, std::initializer_list< std::pair< size_t, size_t > > *initial_list*)

Initializes permutation to have size n, initializes the restrictions with an initializer list of the form {{sigma(a), a},{sigma(b),b},...} meaning a cannot be in location sigma(a), b cannot be in location sigma(b), etc., and computes the sequence.

Parameters

input_n	is the initial size of the permutation
initialize_-restrictions	is the initial set of restrictions

7.53.2 Member Function Documentation

7.53.2.1 template< typename T , typename V , typename SV > V desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::by_pairs, T, V, SV >::first_in_sequence ( ) const

Compute the first instance of a permutation with given restrictions in lexicographic ordering

Returns

the first permutation in lexicographic ordering with the given restrictions

7.53.2.2 template< typename T , typename V , typename SV > void desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::by_pairs, T, V, SV >::insert (const std::pair< size_t, size_t > & *res*)

Inserts a new restriction

Parameters

res	is a new restriction of the form {sigma(a),a}, i.e., a is not in location sigma(a)

7.53.2.3 template< typename T , typename V , typename SV > void desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::by_pairs, T, V, SV >::insert (std::pair< size_t, size_t > && *res*)

Inserts a new restriction

Parameters

res	is a new restriction of the form {sigma(a),a}, i.e., a is not in location sigma(a)

7.53.2.4 template< typename T , typename V , typename SV > void desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::by_pairs, T, V, SV >::insert (std::initializer_list< std::pair< size_t, size_t > > *res*)

Inserts a collection of restrictions in an initializer list

Parameters

res	is a set of new restrictions of the form {{sigma(a),a}, sigma(b),b},...}, i.e., a is not in location sigma(a), b is not in location sigma(b), etc.

7.53.2.5 template<typename T , typename V , typename SV > template<typename InputIterator > void desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::by_pairs, T, V, SV >::insert (InputIterator *start,* InputIterator *stop*)

Inserts a collection of restrictions any collection indexed by an input iterator type

Template Parameters

InputIterator	is any iterator of type input iterator which when dereferenced returns a std::pair<size_t, size_t>

Parameters

res	is a collection of new restrictions of the form {{sigma(a),a}, sigma(b),b},...}, i.e., a is not in location sigma(a), b is not in location sigma(b), etc.

7.53.2.6 template<typename T , typename V , typename SV > V desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::by_pairs, T, V, SV >::last_in_sequence ( ) const

Compute the last instance of a permutation with given restrictions in lexicographic ordering

Returns

the last permutation in lexicographic ordering with the given restrictions

7.53.2.7 template<typename T , typename V , typename SV > bool desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::by_pairs, T, V, SV >::next_in_sequence (V & *v*) const

Given a current state, updates the input to the next state, returns false if next state restarts the sequence.

Parameters

v	is the input state, which is updated to the next state

Returns

whether or not the sequence restarted

7.53.2.8 template<typename T , typename V , typename SV > desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::by_pairs, T, V, SV >::operator bool ( )

conversion to bool, returns true unless the restrictions are too tough and no such set of permutations exist.

Returns

true if there are no elements, false otherwise

7.53.2.9 template<typename T , typename V , typename SV > bool desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::by_pairs, T, V, SV >::previous_in_sequence (V & *v*) const

Given a current state, updates the input to the previous state, returns false if previous state restarts the sequence.

Parameters

v	is the input state, which is updated to the previous state

Returns

whether or not the sequence restarted

7.53.2.10 template<typename T , typename V , typename SV > void desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::by_pairs, T, V, SV >::resize (size_t *input_n*)

Resizes the permutation and then reinitializes with the set of permutations

Parameters

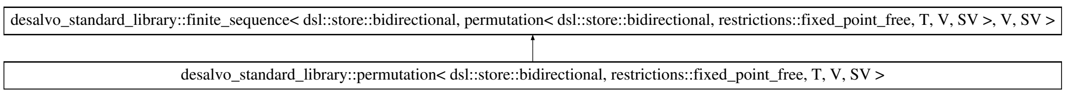
input_n	is the new size of the permutation

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

- DeSalvo Standard Library/desalvo/permutation.h

7.54 desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::fixed_point_free, T, V, SV > Class Template Reference

Inheritance diagram for desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::fixed_point_free, T, V, SV >:



Public Member Functions

- permutation (size_t input_n)
- V first_in_sequence () const
- V last_in_sequence () const
- bool next_in_sequence (V &v) const
- bool previous_in_sequence (V &v) const

7.54.1 Constructor & Destructor Documentation

7.54.1.1 template<typename T , typename V , typename SV > desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::fixed_point_free, T, V, SV >::permutation (size_t *input_n*) [inline]

Initializes permutation to have size n, computes the first and last elements in the sequence, and stores the entire sequence.

Parameters

input_n	is the initial size of the permutation.

7.54.2 Member Function Documentation

7.54.2.1 template<typename T , typename V , typename SV > V desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::fixed_point_free, T, V, SV >::first_in_sequence ( ) const [inline]

Compute the first instance of a permutation with given restrictions in lexicographic ordering

Returns

the first permutation in lexicographic ordering with the given restrictions

7.54.2.2 template<typename T , typename V , typename SV > V desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::fixed_point_free, T, V, SV >::last_in_sequence ( ) const [inline]

Compute the last instance of a permutation with given restrictions in lexicographic ordering

Returns

the last permutation in lexicographic ordering with the given restrictions

7.54.2.3 template<typename T , typename V , typename SV > bool desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::fixed_point_free, T, V, SV >::next_in_sequence (V & *v*) const [inline]

Given a current state, updates the input to the next state, returns false if next state restarts the sequence.

Parameters

v	is the input state, which is updated to the next state

Returns

whether or not the sequence restarted

7.54.2.4 template<typename T , typename V , typename SV > bool desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::fixed_point_free, T, V, SV >::previous_in_sequence (V & *v*) const [inline]

Given a current state, updates the input to the previous state, returns false if next state restarts the sequence.

Parameters

v	is the input state, which is updated to the previous state

Returns

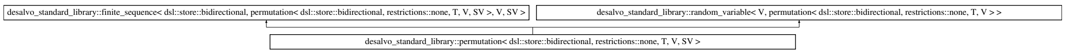
whether or not the sequence restarted

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

- DeSalvo Standard Library/desalvo/permutation.h

7.55 desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::none, T, V, SV > Class Template Reference

Inheritance diagram for desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::none, T, V, SV >:



Public Member Functions

- permutation (size_t input_n)
- V first_in_sequence () const
- V last_in_sequence () const
- bool next_in_sequence (V &v) const
- bool previous_in_sequence (V &v) const
- template<typename URNG >
  V **sample** (URNG &gen=dsl::generator_64)
- template<typename Function , typename URNG >
  V **sample_using** (Function distribution, URNG &gen=dsl::generator_64)

7.55.1 Constructor & Destructor Documentation

7.55.1.1 template<typename T , typename V , typename SV > desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::none, T, V, SV >::permutation (size_t *input_n*) [inline]

Initializes permutation to have size n, computes the first and last elements in the sequence, and stores the entire sequence.

Parameters

input_n	is the initial size of the permutation.

7.55.2 Member Function Documentation

7.55.2.1 template<typename T , typename V , typename SV > V desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::none, T, V, SV >::first_in_sequence ( ) const [inline]

Create {1,2,...,n-1,n}

Returns

the first permutation in lexicographic ordering {1,2,...,n-1,n}

7.55.2.2 template<typename T , typename V , typename SV > V desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::none, T, V, SV >::last_in_sequence ( ) const [inline]

Compute the last instance of a permutation in lexicographic ordering, {n,n-1,...,2,1}

Returns

the last permutation in lexicographic ordering with the given restrictions

7.55.2.3 template<typename T , typename V , typename SV > bool desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::none, T, V, SV >::next_in_sequence (V & *v*) const [inline]

Given a current state, updates the input to the next state, returns false if next state restarts the sequence.

Parameters

v	is the input state, which is updated to the next state

Returns

whether or not the sequence restarted

7.55.2.4 template<typename T , typename V , typename SV > bool desalvo_standard_library::permutation< dsl::store::bidirectional, restrictions::none, T, V, SV >::previous_in_sequence (V & *v*) const [inline]

Given a current state, updates the input to the previous state, returns false if next state restarts the sequence.

Parameters

v	is the input state, which is updated to the previous state

Returns

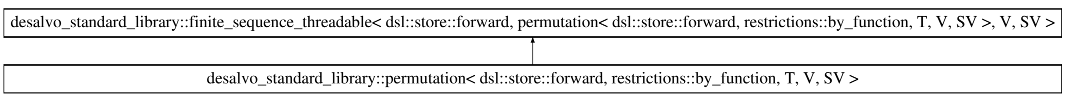
whether or not the sequence restarted

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

- DeSalvo Standard Library/desalvo/permutation.h

7.56 desalvo_standard_library::permutation< dsl::store::forward, restrictions::by_function, T, V, SV > Class Template Reference

Inheritance diagram for desalvo_standard_library::permutation< dsl::store::forward, restrictions::by_function, T, V, SV >:



Public Member Functions

- permutation (size_t input_n)
- permutation (size_t input_n, std::function< bool(V &)> restriction_function)
- void replace_restriction_function (std::function< bool(V &)> new_restriction_function)
- void resize (size_t n)
- V first_in_sequence () const
- V first_in_sequence (size_t i) const
- bool next_in_sequence (V &v) const
- operator bool ()

7.56.1 Constructor & Destructor Documentation

7.56.1.1 template<typename T , typename V , typename SV > desalvo_standard_library::permutation< dsl::store::forward, restrictions::by_function, T, V, SV >::permutation (size_t *input_n*)

Initializes permutation to have size n, with no restrictions, computes the first element in the sequence, and stores the entire sequence.

Parameters

input_n	is the initial size of the permutation.

7.56.1.2 template<typename T , typename V , typename SV > desalvo_standard_library::permutation< dsl::store::forward, restrictions::by_function, T, V, SV >::permutation (size_t *input_n,* std::function< bool(V &)> *initialize_restrictions*)

Initializes permutation to have size n, initializes the restrictions, and computes the sequence.

Parameters

input_n	is the initial size of the permutation
initialize_-restrictions	is the initial set of restrictions

7.56.2 Member Function Documentation

7.56.2.1 template<typename T , typename V , typename SV > V desalvo_standard_library::permutation< dsl::store::forward, restrictions::by_function, T, V, SV >::first_in_sequence ( ) const

Compute the last instance of a permutation with given restrictions in lexicographic ordering

Returns

the last permutation in lexicographic ordering with the given restrictions

7.56.2.2 template<typename T , typename V , typename SV > V desalvo_standard_library::permutation< dsl::store::forward, restrictions::by_function, T, V, SV >::first_in_sequence (size_t *i*) const

Compute the last instance of a permutation with given restrictions in lexicographic ordering

Returns

the last permutation in lexicographic ordering with the given restrictions

7.56.2.3 template<typename T , typename V , typename SV > bool desalvo_standard_library::permutation< dsl::store::forward, restrictions::by_function, T, V, SV >::next_in_sequence (V & *v*) const

Given a current state, updates the input to the next state, returns false if next state restarts the sequence.

Parameters

v	is the input state, which is updated to the next state

Returns

whether or not the sequence restarted

7.56.2.4 template< typename T , typename V , typename SV > desalvo_standard_library::permutation< dsl::store::forward, restrictions::by_function, T, V, SV >::operator bool ( )

conversion to bool, returns true unless the restrictions are too tough and no such set of permutations exist.

Returns

true if there are no elements, false otherwise

7.56.2.5 template< typename T , typename V , typename SV > void desalvo_standard_library::permutation< dsl::store::forward, restrictions::by_function, T, V, SV >::replace_restriction_function (std::function< bool(V &)> *new_restriction_function*)

replaces restriction function with the one input, recalculates the set of objects

Parameters

new_restriction-_function	is the new function to test against violations

7.56.2.6 template< typename T , typename V , typename SV > void desalvo_standard_library::permutation< dsl::store::forward, restrictions::by_function, T, V, SV >::resize (size_t *input_n*)

Resizes the permutation and then reinitializes with the set of permutations

Parameters

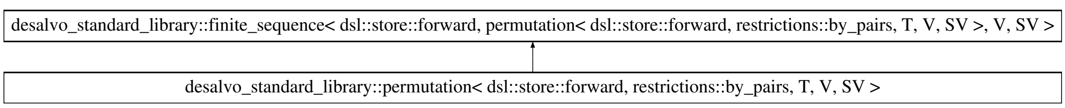
input_n	is the new size of the permutation

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

- DeSalvo Standard Library/desalvo/permutation.h

7.57 desalvo_standard_library::permutation< dsl::store::forward, restrictions::by_pairs, T, V, SV > Class Template Reference

Inheritance diagram for desalvo_standard_library::permutation< dsl::store::forward, restrictions::by_pairs, T, V, SV >:



Public Member Functions

- permutation (size_t input_n)
- permutation (size_t input_n, const std::set< std::pair< size_t, size_t > > &initialize_restrictions)

- permutation (size_t input_n, std::initializer_list< std::pair< size_t, size_t > > initial_list)
- void resize (size_t n)
- V first_in_sequence () const
- bool next_in_sequence (V &v) const
- void insert (const std::pair< size_t, size_t > &res)
- void insert (std::pair< size_t, size_t > &&res)
- void insert (std::initializer_list< std::pair< size_t, size_t > > res)
- template<typename InputIterator >
  void insert (InputIterator start, InputIterator stop)
- void clear ()
- operator bool ()

7.57.1 Constructor & Destructor Documentation

7.57.1.1 template<typename T , typename V , typename SV > desalvo_standard_library::permutation< dsl::store::forward, restrictions::by_pairs, T, V, SV >::permutation (size_t *input_n*)

Initializes permutation to have size n, computes the first element in the sequence, and stores the entire sequence.

Parameters

input_n	is the initial size of the permutation.

7.57.1.2 template<typename T , typename V , typename SV > desalvo_standard_library::permutation< dsl::store::forward, restrictions::by_pairs, T, V, SV >::permutation (size_t *input_n*, const std::set< std::pair< size_t, size_t > > & *initialize_restrictions*)

Initializes permutation to have size n, initializes the restrictions, and computes the sequence.

Parameters

input_n	is the initial size of the permutation
initialize_-restrictions	is the initial set of restrictions

7.57.1.3 template<typename T , typename V , typename SV > desalvo_standard_library::permutation< dsl::store::forward, restrictions::by_pairs, T, V, SV >::permutation (size_t *input_n*, std::initializer_list< std::pair< size_t, size_t > > *initial_list*)

Initializes permutation to have size n, initializes the restrictions with an initializer list of the form {{sigma(a), a},{sigma(b),b},...} meaning a cannot be in location sigma(a), b cannot be in location sigma(b), etc., and computes the sequence.

Parameters

input_n	is the initial size of the permutation
initialize_-restrictions	is the initial set of restrictions

7.57.2 Member Function Documentation

7.57.2.1 template< typename T , typename V , typename SV > void desalvo_standard_library::permutation< dsl::store::forward, restrictions::by_pairs, T, V, SV >::clear ( ) [inline]

Clears all restrictions, does NOT recompute.

7.57.2.2 template< typename T , typename V , typename SV > V desalvo_standard_library::permutation< dsl::store::forward, restrictions::by_pairs, T, V, SV >::first_in_sequence ( ) const

Compute the last instance of a permutation with given restrictions in lexicographic ordering

Returns

the last permutation in lexicographic ordering with the given restrictions

7.57.2.3 template< typename T , typename V , typename SV > void desalvo_standard_library::permutation< dsl::store::forward, restrictions::by_pairs, T, V, SV >::insert (const std::pair< size_t, size_t > & *res*)

Inserts a new restriction

Parameters

res	is a new restriction of the form {sigma(a),a}, i.e., a is not in location sigma(a)

7.57.2.4 template< typename T , typename V , typename SV > void desalvo_standard_library::permutation< dsl::store::forward, restrictions::by_pairs, T, V, SV >::insert (std::pair< size_t, size_t > && *res*)

Inserts a new restriction

Parameters

res	is a new restriction of the form {sigma(a),a}, i.e., a is not in location sigma(a)

7.57.2.5 template< typename T , typename V , typename SV > void desalvo_standard_library::permutation< dsl::store::forward, restrictions::by_pairs, T, V, SV >::insert (std::initializer_list< std::pair< size_t, size_t > > *res*)

Inserts a collection of restrictions in an initializer list

Parameters

res	is a set of new restrictions of the form {{sigma(a),a}, sigma(b),b},...}, i.e., a is not in location sigma(a), b is not in location sigma(b), etc.

7.57.2.6 template< typename T , typename V , typename SV > template< typename InputIterator > void desalvo_standard_library::permutation< dsl::store::forward, restrictions::by_pairs, T, V, SV >::insert (InputIterator *start,* InputIterator *stop*)

Inserts a collection of restrictions any collection indexed by an input iterator type

Template Parameters

InputIterator	is any iterator of type input iterator which when dereferenced returns a std::pair<size_t, size_t>

Parameters

res	is a collection of new restrictions of the form {{sigma(a),a}, sigma(b),b},...}, i.e., a is not in location sigma(a), b is not in location sigma(b), etc.

7.57.2.7 template<typename T , typename V , typename SV > bool desalvo_standard_library::permutation< dsl::store::forward, restrictions::by_pairs, T, V, SV >::next_in_sequence (V & *v*) const

Given a current state, updates the input to the next state, returns false if next state restarts the sequence.

Parameters

v	is the input state, which is updated to the next state

Returns

whether or not the sequence restarted

7.57.2.8 template<typename T , typename V , typename SV > desalvo_standard_library::permutation< dsl::store::forward, restrictions::by_pairs, T, V, SV >::operator bool ( )

conversion to bool, returns true unless the restrictions are too tough and no such set of permutations exist.

Returns

true if there are no elements, false otherwise

7.57.2.9 template<typename T , typename V , typename SV > void desalvo_standard_library::permutation< dsl::store::forward, restrictions::by_pairs, T, V, SV >::resize (size_t *input_n*)

Resizes the permutation and then reinitializes with the set of permutations

Parameters

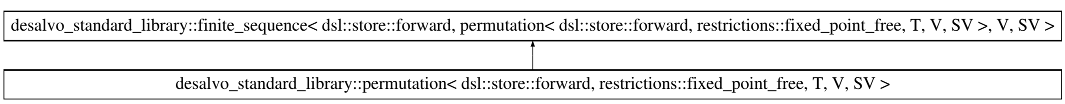
input_n	is the new size of the permutation

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

- DeSalvo Standard Library/desalvo/permutation.h

7.58 desalvo_standard_library::permutation< dsl::store::forward, restrictions::fixed_point_free, T, V, SV > Class Template Reference

Inheritance diagram for desalvo_standard_library::permutation< dsl::store::forward, restrictions::fixed_point_free, T, V, SV >:



Public Member Functions

- permutation (size_t input_n)
- V first_in_sequence () const
- bool next_in_sequence (V &v) const

7.58.1 Constructor & Destructor Documentation

7.58.1.1 template<typename T , typename V , typename SV > desalvo_standard_library::permutation< dsl::store::forward, restrictions::fixed_point_free, T, V, SV >::permutation (size_t *input_n*) [inline]

Initializes permutation to have size n, computes the first element in the sequence.

Parameters

input_n	is the initial size of the permutation.

7.58.2 Member Function Documentation

7.58.2.1 template<typename T , typename V , typename SV > V desalvo_standard_library::permutation< dsl::store::forward, restrictions::fixed_point_free, T, V, SV >::first_in_sequence ( ) const [inline]

Compute the last instance of a permutation with given restrictions in lexicographic ordering

Returns

the last permutation in lexicographic ordering with the given restrictions

7.58.2.2 template<typename T , typename V , typename SV > bool desalvo_standard_library::permutation< dsl::store::forward, restrictions::fixed_point_free, T, V, SV >::next_in_sequence (V & *v*) const [inline]

Given a current state, updates the input to the next state, returns false if next state restarts the sequence.

Parameters

v	is the input state, which is updated to the next state

Returns

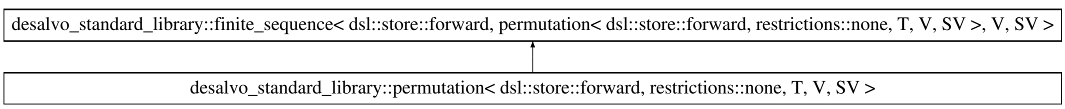
whether or not the sequence restarted

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

- DeSalvo Standard Library/desalvo/permutation.h

7.59 desalvo_standard_library::permutation< dsl::store::forward, restrictions::none, T, V, SV > Class Template Reference

Inheritance diagram for desalvo_standard_library::permutation< dsl::store::forward, restrictions::none, T, V, SV >:



Public Member Functions

- permutation (size_t input_n)
- V first_in_sequence () const
- bool next_in_sequence (V &v) const

7.59.1 Constructor & Destructor Documentation

7.59.1.1 template<typename T , typename V , typename SV > desalvo_standard_library::permutation< dsl::store::forward, restrictions::none, T, V, SV >::permutation (size_t *input_n*) [inline]

Initializes permutation to have size n, computes the first element in the sequence.

Parameters

input_n	is the initial size of the permutation.

7.59.2 Member Function Documentation

7.59.2.1 template<typename T , typename V , typename SV > V desalvo_standard_library::permutation< dsl::store::forward, restrictions::none, T, V, SV >::first_in_sequence ( ) const [inline]

Create {1,2,...,n-1,n}

Returns

the first permutation in lexicographic ordering {1,2,...,n-1,n}

7.59.2.2 template<typename T , typename V , typename SV > bool desalvo_standard_library::permutation< dsl::store::forward, restrictions::none, T, V, SV >::next_in_sequence (V & *v*) const [inline]

Given a current state, updates the input to the next state, returns false if next state restarts the sequence.

Parameters

v	is the input state, which is updated to the next state

Returns

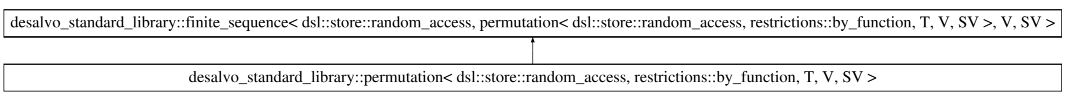
whether or not the sequence restarted

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

- DeSalvo Standard Library/desalvo/permutation.h

7.60 desalvo_standard_library::permutation< dsl::store::random_access, restrictions::by_function, T, V, SV > Class Template Reference

Inheritance diagram for desalvo_standard_library::permutation< dsl::store::random_access, restrictions::by_function, T, V, SV >:



Public Member Functions

- permutation (size_t input_n)
- permutation (size_t input_n, std::function< bool(V &)> restriction_function)
- void replace_restriction_function (std::function< bool(V &)> new_restriction_function)
- void resize (size_t n)
- V first_in_sequence () const
- bool next_in_sequence (V &v) const
- operator bool ()

7.60.1 Constructor & Destructor Documentation

7.60.1.1 template<typename T , typename V , typename SV > **desalvo_standard_library::permutation< dsl::store::random_access, restrictions::by_function, T, V, SV >::permutation** (size_t *input_n*)

Initializes permutation to have size n, computes the first element in the sequence, and stores the entire sequence.

Parameters

input_n	is the initial size of the permutation.

7.60.1.2 template<typename T , typename V , typename SV > **desalvo_standard_library::permutation< dsl::store::random_access, restrictions::by_function, T, V, SV >::permutation** (size_t *input_n,* std::function< bool(V &)> *initialize_restrictions*)

Initializes permutation to have size n, initializes the restrictions, and computes the sequence.

Parameters

input_n	is the initial size of the permutation
initialize_-restrictions	is the initial set of restrictions

7.60.2 Member Function Documentation

7.60.2.1 template<typename T , typename V , typename SV > V **desalvo_standard_library::permutation< dsl::store::random_access, restrictions::by_function, T, V, SV >::first_in_sequence** ( ) const

Finds and returns the first in lexicographic ordering of the sequence with restrictions

Returns

the first in lexicographic ordering of the sequence with restrictions

7.60.2.2 template<typename T , typename V , typename SV > bool **desalvo_standard_library::permutation< dsl::store::random_access, restrictions::by_function, T, V, SV >::next_in_sequence** (V & *v*) const

Given a current state, updates the input to the next state, returns false if next state restarts the sequence.

Parameters

v	is the input state, which is updated to the next state

Returns

whether or not the sequence restarted

7.60.2.3 template<typename T , typename V , typename SV > desalvo_standard_library::permutation< dsl::store::random_access, restrictions::by_function, T, V, SV >::operator bool ( )

conversion to bool, returns true unless the restrictions are too tough and no such set of permutations exist.

Returns

true if there are no elements, false otherwise

7.60.2.4 template<typename T , typename V , typename SV > void desalvo_standard_library::permutation< dsl::store::random_access, restrictions::by_function, T, V, SV >::replace_restriction_function (std::function< bool(V &)> *new_restriction_function*)

replaces restriction function with the one input, recalculates the set of objects

Parameters

new_restriction-_function	is the new function to test against violations

7.60.2.5 template<typename T , typename V , typename SV > void desalvo_standard_library::permutation< dsl::store::random_access, restrictions::by_function, T, V, SV >::resize (size_t *input_n*)

Resizes the permutation and then reinitializes with the set of permutations

Parameters

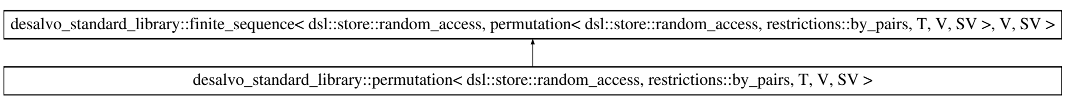
input_n	is the new size of the permutation

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

- DeSalvo Standard Library/desalvo/permutation.h

7.61 desalvo_standard_library::permutation< dsl::store::random_access, restrictions::by_pairs, T, V, SV > Class Template Reference

Inheritance diagram for desalvo_standard_library::permutation< dsl::store::random_access, restrictions::by_pairs, T, V, SV >:



Public Member Functions

- permutation (size_t input_n)
- permutation (size_t input_n, const std::set< std::pair< size_t, size_t > > &initialize_restrictions)

- permutation (size_t input_n, std::initializer_list< std::pair< size_t, size_t > > initial_list)
- void resize (size_t n)
- V first_in_sequence () const
- bool next_in_sequence (V &v) const
- void insert (const std::pair< size_t, size_t > &res)
- void insert (std::pair< size_t, size_t > &&res)
- void insert (std::initializer_list< std::pair< size_t, size_t > > res)
- template<typename InputIterator >
  void insert (InputIterator start, InputIterator stop)
- void **clear** ()
- operator bool ()

7.61.1 Constructor & Destructor Documentation

7.61.1.1 template<typename T , typename V , typename SV > desalvo_standard_library::permutation< dsl::store::random_access, restrictions::by_pairs, T, V, SV >::permutation (size_t *input_n*)

Initializes permutation to have size n, computes the first element in the sequence, and stores the entire sequence.

Parameters

input_n	is the initial size of the permutation.

7.61.1.2 template<typename T , typename V , typename SV > desalvo_standard_library::permutation< dsl::store::random_access, restrictions::by_pairs, T, V, SV >::permutation (size_t *input_n*, const std::set< std::pair< size_t, size_t > > & *initialize_restrictions*)

Initializes permutation to have size n, initializes the restrictions, and computes the sequence.

Parameters

input_n	is the initial size of the permutation
initialize_-restrictions	is the initial set of restrictions

7.61.1.3 template<typename T , typename V , typename SV > desalvo_standard_library::permutation< dsl::store::random_access, restrictions::by_pairs, T, V, SV >::permutation (size_t *input_n*, std::initializer_list< std::pair< size_t, size_t > > *initial_list*)

Initializes permutation to have size n, initializes the restrictions with an initializer list of the form {{sigma(a), a},{sigma(b),b},...} meaning a cannot be in location sigma(a), b cannot be in location sigma(b), etc., and computes the sequence.

Parameters

input_n	is the initial size of the permutation
initialize_-restrictions	is the initial set of restrictions

7.61.2 Member Function Documentation

7.61.2.1 template<typename T , typename V , typename SV > V desalvo_standard_library::permutation< dsl::store::random_access, restrictions::by_pairs, T, V, SV >::first_in_sequence ( ) const

Finds and returns the first in lexicographic ordering of the sequence with restrictions

Returns

the first in lexicographic ordering of the sequence with restrictions

7.61.2.2 template<typename T , typename V , typename SV > void desalvo_standard_library::permutation< dsl::store::random_access, restrictions::by_pairs, T, V, SV >::insert (const std::pair< size_t, size_t > & *res*)

Inserts a new restriction

Parameters

res	is a new restriction of the form {sigma(a),a}, i.e., a is not in location sigma(a)

7.61.2.3 template<typename T , typename V , typename SV > void desalvo_standard_library::permutation< dsl::store::random_access, restrictions::by_pairs, T, V, SV >::insert (std::pair< size_t, size_t > && *res*)

Inserts a new restriction

Parameters

res	is a new restriction of the form {sigma(a),a}, i.e., a is not in location sigma(a)

7.61.2.4 template<typename T , typename V , typename SV > void desalvo_standard_library::permutation< dsl::store::random_access, restrictions::by_pairs, T, V, SV >::insert (std::initializer_list< std::pair< size_t, size_t > > *res*)

Inserts a collection of restrictions in an initializer list

Parameters

res	is a set of new restrictions of the form {{sigma(a),a}, sigma(b),b},...}, i.e., a is not in location sigma(a), b is not in location sigma(b), etc.

7.61.2.5 template<typename T , typename V , typename SV > template<typename InputIterator > void desalvo_standard_library::permutation< dsl::store::random_access, restrictions::by_pairs, T, V, SV >::insert (InputIterator *start,* InputIterator *stop*)

Inserts a collection of restrictions any collection indexed by an input iterator type

Template Parameters

InputIterator	is any iterator of type input iterator which when dereferenced returns a std::pair<size_t, size_t>

Parameters

res	is a collection of new restrictions of the form {{sigma(a),a}, sigma(b),b},...}, i.e., a is not in location sigma(a), b is not in location sigma(b), etc.

7.61.2.6 template<typename T , typename V , typename SV > bool desalvo_standard_library::permutation< dsl::store::random_access, restrictions::by_pairs, T, V, SV >::next_in_sequence (V & *v*) const

Given a current state, updates the input to the next state, returns false if next state restarts the sequence.

Parameters

v	is the input state, which is updated to the next state

Returns

whether or not the sequence restarted

7.61.2.7 template<typename T , typename V , typename SV > desalvo_standard_library::permutation< dsl::store::random_access, restrictions::by_pairs, T, V, SV >::operator bool ( )

conversion to bool, returns true unless the restrictions are too tough and no such set of permutations exist.

Returns

true if there are no elements, false otherwise

7.61.2.8 template<typename T , typename V , typename SV > void desalvo_standard_library::permutation< dsl::store::random_access, restrictions::by_pairs, T, V, SV >::resize (size_t *input_n*)

Resizes the permutation and then reinitializes with the set of permutations

Parameters

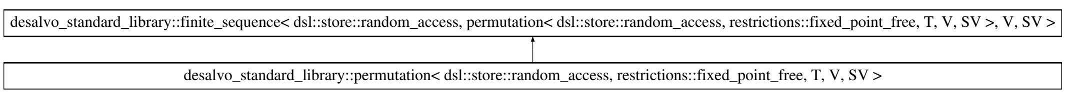
input_n	is the new size of the permutation

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

- DeSalvo Standard Library/desalvo/permutation.h

7.62 desalvo_standard_library::permutation< dsl::store::random_access, restrictions::fixed_point_free, T, V, SV > Class Template Reference

Inheritance diagram for desalvo_standard_library::permutation< dsl::store::random_access, restrictions::fixed_point_free, T, V, SV >:



Public Member Functions

- permutation (size_t input_n)
- V first_in_sequence () const
- bool next_in_sequence (V &v) const

7.62.1 Constructor & Destructor Documentation

7.62.1.1 template< typename T , typename V , typename SV > desalvo_standard_library::permutation< dsl::store::random_access, restrictions::fixed_point_free, T, V, SV >::permutation (size_t *input_n*) [inline]

Initializes permutation to have size n, computes the first element in the sequence, and stores the entire sequence.

Parameters

input_n	is the initial size of the permutation.

7.62.2 Member Function Documentation

7.62.2.1 template< typename T , typename V , typename SV > V desalvo_standard_library::permutation< dsl::store::random_access, restrictions::fixed_point_free, T, V, SV >::first_in_sequence ( ) const [inline]

Compute the first instance of a permutation with given restrictions in lexicographic ordering

Returns

the first permutation in lexicographic ordering with the given restrictions

7.62.2.2 template< typename T , typename V , typename SV > bool desalvo_standard_library::permutation< dsl::store::random_access, restrictions::fixed_point_free, T, V, SV >::next_in_sequence (V & *v*) const [inline]

Compute the last instance of a permutation with given restrictions in lexicographic ordering

Returns

the last permutation in lexicographic ordering with the given restrictions Given a current state, updates the input to the next state, returns false if next state restarts the sequence.

Parameters

v	is the input state, which is updated to the next state

Returns

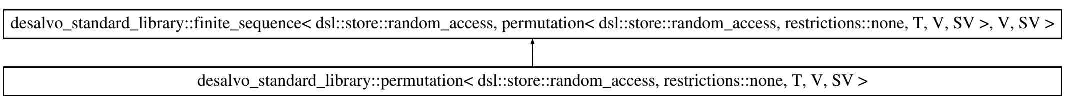
whether or not the sequence restarted

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

- DeSalvo Standard Library/desalvo/permutation.h

7.63 desalvo_standard_library::permutation< dsl::store::random_access, restrictions::none, T, V, SV > Class Template Reference

Inheritance diagram for desalvo_standard_library::permutation< dsl::store::random_access, restrictions::none, T, V, SV >:



Public Member Functions

- permutation (size_t input_n)
- V first_in_sequence () const
- bool next_in_sequence (V &v) const

7.63.1 Constructor & Destructor Documentation

7.63.1.1 template<typename T , typename V , typename SV > desalvo_standard_library::permutation< dsl::store::random_access, restrictions::none, T, V, SV >::permutation (size_t *input_n*) [inline]

Initializes permutation to have size n, computes the first element in the sequence, and stores the entire sequence.

Parameters

input_n	is the initial size of the permutation.

7.63.2 Member Function Documentation

7.63.2.1 template<typename T , typename V , typename SV > V desalvo_standard_library::permutation< dsl::store::random_access, restrictions::none, T, V, SV >::first_in_sequence ( ) const [inline]

creates {1,2,...,n}

Returns

the first permutation in lexicographic ordering with the given restrictions

7.63.2.2 template<typename T , typename V , typename SV > bool desalvo_standard_library::permutation< dsl::store::random_access, restrictions::none, T, V, SV >::next_in_sequence (V & *v*) const [inline]

Given a current state, updates the input to the next state, returns false if next state restarts the sequence.

Parameters

v	is the input state, which is updated to the next state

Returns

whether or not the sequence restarted

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

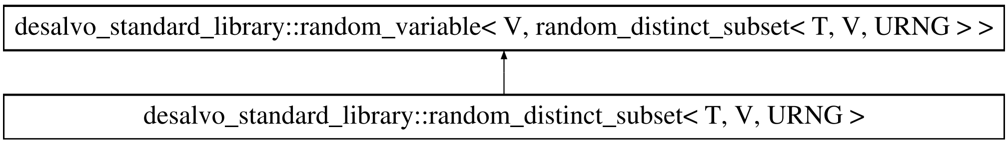
- DeSalvo Standard Library/desalvo/permutation.h

7.64 desalvo_standard_library::random_distinct_subset< T, V, URNG > Class Template Reference

Generates a subset of size k from {1,2,...,n}.

```
#include <statistics.h>
```

Inheritance diagram for desalvo_standard_library::random_distinct_subset< T, V, URNG >:



Public Member Functions

- random_distinct_subset (T input_n, T input_k)
- random_distinct_subset ()
- V operator() (URNG &gen=generator_64)
- void set_param (T input_n, T input_k)

7.64.1 Detailed Description

template<typename T, typename V = std::vector<T>, typename URNG = std::mt19937_64>class desalvo_standard_library::random_distinct_subset< T, V, URNG >

Generates a subset of size k from {1,2,...,n}.

Many times I was wanting to generate a distinct set of indices from among {1,2,...,n}. For example, placing rooks on a chess board the locations can be indexed by {1,2,...,n choose 2}, and if there are k rooks then we can generate a distinct subset of size k from this set and then map these indices to coordinates on a board.

7.64.2 Constructor & Destructor Documentation

7.64.2.1 template<typename T , typename V = std::vector<T>, typename URNG = std::mt19937_64> desalvo_standard_library::random_distinct_subset< T, V, URNG >::random_distinct_subset (T *input_n*, T *input_k*) [inline]

Initialize the set and subset size

Parameters

input_n	is the entire set of possible indices
input_k	is the size of the set of distinct elements

7.64.2.2 template<typename T , typename V = std::vector<T>, typename URNG = std::mt19937_64> desalvo_standard_library::random_distinct_subset< T, V, URNG >::random_distinct_subset ( ) [inline]

Initialize the set and subset size to 0 and 0 by default, empty sets.

7.64.3 Member Function Documentation

7.64.3.1 template<typename T , typename V = std::vector<T>, typename URNG = std::mt19937_64> V desalvo_standard_library::random_distinct_subset< T, V, URNG >::operator() (URNG & *gen* = generator_64) [inline]

Overloaded operator for use with CRTP

Parameters

gen	is the random number generator, by default 64 bits

Returns

a random subset of k distinct numbers from the set {1,2,...,n}

7.64.3.2 template<typename T , typename V = std::vector<T>, typename URNG = std::mt19937_64> void desalvo_standard_library::random_distinct_subset< T, V, URNG >::set_param (T *input_n*, T *input_k*) [inline]

Resets the parameters

Parameters

input_n	is the new entire set of possible indices
input_k	is the new size of the set of distinct elements

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

- DeSalvo Standard Library/desalvo/statistics.h

7.65 desalvo_standard_library::random_variable< T, Derived, URNG > Class Template Reference

Public Member Functions

- template<typename V = std::vector<T>>
  V iid_sample (size_t m, URNG &gen=generator_64)
- template<typename F = double>
  F estimate_mean (size_t m, URNG &gen=generator_64)

7.65.1 Member Function Documentation

7.65.1.1 template<typename T, typename Derived, typename URNG = std::mt19937_64> template<typename F = double> F desalvo_standard_library::random_variable< T, Derived, URNG >::estimate_mean (size_t *m*, URNG & *gen* = generator_64) [inline]

Estimates the average using iid samples. Return type F can be different from T

Parameters

m	is the number of samples
gen	is the random number generator, by default 64-bit

Returns

an average of generated values by static_cast to element of type F

7.65.1.2 template<typename T, typename Derived, typename URNG = std::mt19937_64> template<typename V = std::vector<T>> V desalvo_standard_library::random_variable< T, Derived, URNG >::iid_sample (size_t *m*, URNG & *gen* = `generator_64`) `[inline]`

Generator for iid samples. The container for output must be constructable using size_t input parameter.

Parameters

m	is the number of samples
gen	is the random number generator, by default 64-bit

Returns

an iid sample stored in container of template type V=std::vector<T>.

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

- DeSalvo Standard Library/desalvo/statistics.h

7.66 RandomVariable Class Reference

CRTP base class for random objects.

`#include <statistics.h>`

7.66.1 Detailed Description

CRTP base class for random objects.

CRTP base class for objects that can be generated randomly.

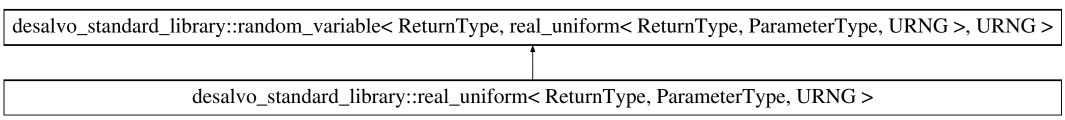
Requires: operator()(URNG& gen) overloaded for class Derived

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

- DeSalvo Standard Library/desalvo/statistics.h

7.67 desalvo_standard_library::real_uniform< ReturnType, ParameterType, URNG > Class Template Reference

Inheritance diagram for desalvo_standard_library::real_uniform< ReturnType, ParameterType, URNG >:



Public Member Functions

- real_uniform (ParameterType a, ParameterType b)

- ReturnType operator() (URNG &gen=generator_64)
- template<typename F = double>
  F mean ()

7.67.1 Constructor & Destructor Documentation

7.67.1.1 template<typename ReturnType = double, typename ParameterType = ReturnType, typename URNG = std::mt19937_64> desalvo_standard_library::real_uniform< ReturnType, ParameterType, URNG >::real_uniform (ParameterType *a*, ParameterType *b*) [inline]

Constructs a random variable for random values in {a,a+1,...,b-1,b}

Parameters

a	is the lower bound
b	is the upper bound

7.67.2 Member Function Documentation

7.67.2.1 template<typename ReturnType = double, typename ParameterType = ReturnType, typename URNG = std::mt19937_64> template<typename F = double> F desalvo_standard_library::real_uniform< ReturnType, ParameterType, URNG >::mean ( ) [inline]

Returns the expected value of the random variable ReturnType must have operator+(ReturnType, ReturnType) defined, and the return type of operator+ must be castable to F

Returns

(lower+upper)/2

7.67.2.2 template<typename ReturnType = double, typename ParameterType = ReturnType, typename URNG = std::mt19937_64> ReturnType desalvo_standard_library::real_uniform< ReturnType, ParameterType, URNG >::operator() (URNG & *gen* = generator_64) [inline]

Generates random value from distribution using the std distributions.

Parameters

gen	is the random number generator, by default 64-bit.

Returns

random element

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

- DeSalvo Standard Library/desalvo/statistics.h

7.68 RealUniform Class Reference

Uniform over an interval [a,b].

`#include <statistics.h>`

7.68.1 Detailed Description

Uniform over an interval [a,b].

Inherits from RandomVariable so as long as operator()(URNG& gen) is overloaded to return a random value we can invoke its member functions.

Store a lower and upper bound denoting the smallest and largest values capable of being generated.

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

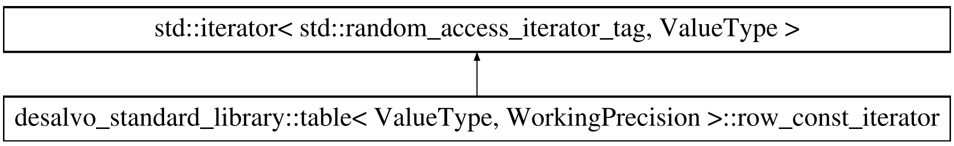
- DeSalvo Standard Library/desalvo/statistics.h

7.69 desalvo_standard_library::table< ValueType, WorkingPrecision >::row_const_iterator Class Reference

Random Access const_iterator for Rows, muentry.

```
#include <table.h>
```

Inheritance diagram for desalvo_standard_library::table< ValueType, WorkingPrecision >::row_const_iterator:



Public Member Functions

- row_const_iterator (const table *initial_mat=nullptr, int initial_row=0, int initial_col=0)
- **row_const_iterator** (const row_const_iterator &other)
- void **swap** (row_const_iterator &other)
- row_const_iterator & **operator=** (row_const_iterator to_copy)
- row_const_iterator & operator++ ()
- row_const_iterator operator++ (int)
- row_const_iterator & operator+= (int col)
- row_const_iterator operator+ (int col) const
- row_const_iterator & operator-- ()
- row_const_iterator operator-- (int)
- row_const_iterator & operator-= (int col)
- row_const_iterator operator- (int col) const
- int operator- (const row_const_iterator &p2) const
- ValueType & operator* () const
- ValueType & operator-> () const
- ValueType & operator[] (int n) const

Friends

- bool operator== (const table::row_const_iterator &lhs, const table::row_const_iterator &rhs)
- bool operator< (const table::row_const_iterator &lhs, const table::row_const_iterator &rhs)

7.69.1 Detailed Description

template<typename ValueType = double, typename WorkingPrecision = long double>class desalvo_standard_library::table< ValueType, WorkingPrecision >::row_const_iterator

Random Access const_iterator for Rows, muentry.

This class is designed to be a RANDOM ACCESS const_iterator for a given row of the entry.

The row is modifiable so that it can change which row it is pointing to. The column is modified along with the pointer so that it is easier to keep track of bounds.

```
#include <iostream>
#include <vector>
#include "std_cout.h"
#include "desalvo/table.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Make a vector of values
std::vector<int> v(100);

// initialize values to 0,1,2,...,99
std::iota(std::begin(v), std::end(v), 0);

// Initialize 10x10 table using 10 numbers for each row from v
dsl::table<int> t(10,10,std::begin(v));

// print out the table of values first
std::cout << "t: \n" << t << std::endl << std::endl;

// Iterate through every fourth column, squaring each element
for(auto j = 0; j<10; j += 4) {
// Square each value
std::for_each(t.begin_column(j), t.end_column(j), [](int& a) { a *= a; });
}

std::cout << "Printing every other column ... \n";
// Iterate through every other row, print it out
for(auto i = 0; i<10; i += 2) {
// Print every other row
std::cout << "{";
std::for_each(t.cbegin_row(i), t.cend_row(i), [](int a) { std::cout << a<<","; });
std::cout << "}\n\n";
}

// Sort each column...no good reason, just want to demonstrate that the iterators work even for algorithms
       which require random access iterators.  Note that begin_column and end_column return objects, not raw
       pointers, since raw pointers will not observe the desired behavior for the ROW-MAJOR table format.
for(auto i = 0;i<10;++i)
std::sort(t.begin_column(i), t.end_column(i));

std::cout << "After all that, sort elements in each column, and print t again:\n";
std::cout << t << std::endl;

return 0;
}
```

Should produce output

```
t:
{{0,1,2,3,4,5,6,7,8,9},
{10,11,12,13,14,15,16,17,18,19},
{20,21,22,23,24,25,26,27,28,29},
{30,31,32,33,34,35,36,37,38,39},
{40,41,42,43,44,45,46,47,48,49},
{50,51,52,53,54,55,56,57,58,59},
{60,61,62,63,64,65,66,67,68,69},
{70,71,72,73,74,75,76,77,78,79},
{80,81,82,83,84,85,86,87,88,89},
{90,91,92,93,94,95,96,97,98,99}}

Printing every other column ...
{0,1,2,3,16,5,6,7,64,9,}

{400,21,22,23,576,25,26,27,784,29,}

{1600,41,42,43,1936,45,46,47,2304,49,}
```

```
{3600,61,62,63,4096,65,66,67,4624,69,}

{6400,81,82,83,7056,85,86,87,7744,89,}

After all that, sort elements in each column, and print t again:
{{0,1,2,3,16,5,6,7,64,9},
{100,11,12,13,196,15,16,17,324,19},
{400,21,22,23,576,25,26,27,784,29},
{900,31,32,33,1156,35,36,37,1444,39},
{1600,41,42,43,1936,45,46,47,2304,49},
{2500,51,52,53,2916,55,56,57,3364,59},
{3600,61,62,63,4096,65,66,67,4624,69},
{4900,71,72,73,5476,75,76,77,6084,79},
{6400,81,82,83,7056,85,86,87,7744,89},
{8100,91,92,93,8836,95,96,97,9604,99}}
```

7.69.2 Constructor & Destructor Documentation

7.69.2.1 template<typename ValueType = double, typename WorkingPrecision = long double> desalvo_standard_library::table< ValueType, WorkingPrecision >::row_const_iterator::row_const_iterator (const table * *initial_mat* = nullptr, int *initial_row* = 0, int *initial_col* = 0) [inline]

Construct by table object

Parameters

T	is the table object with data
r	is the row number.
col	is the column

7.69.3 Member Function Documentation

7.69.3.1 template<typename ValueType = double, typename WorkingPrecision = long double> ValueType& desalvo_standard_library::table< ValueType, WorkingPrecision >::row_const_iterator::operator* ( ) const [inline]

Dereference operator

Returns

the value the const_iterator points to.

7.69.3.2 template<typename ValueType = double, typename WorkingPrecision = long double> row_const_iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::row_const_iterator::operator+ (int *col*) const [inline]

Increment operator

Parameters

col	is the number of elements to increment over, can be negative

Returns

a new iterator referring to the incremented value

7.69.3.3 template< typename ValueType = double, typename WorkingPrecision = long double > row_const_iterator& desalvo_standard_library::table< ValueType, WorkingPrecision >::row_const_iterator::operator++ ( ) [inline]

Standard prefix ++ operator

Returns

reference to self after the increment.

7.69.3.4 template< typename ValueType = double, typename WorkingPrecision = long double > row_const_iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::row_const_iterator::operator++ (int) [inline]

Postfix ++ operator, increments iterator to the next element

Parameters

unused	is an unused parameter

Returns

an iterator referring to the element before the increment

7.69.3.5 template< typename ValueType = double, typename WorkingPrecision = long double > row_const_iterator& desalvo_standard_library::table< ValueType, WorkingPrecision >::row_const_iterator::operator+= (int *col*) [inline]

Increment equals operator

Parameters

col	is the number of elements to increment over, can be negative

Returns

reference to the iterator for chaining

7.69.3.6 template< typename ValueType = double, typename WorkingPrecision = long double > row_const_iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::row_const_iterator::operator- (int *col*) const [inline]

Decrement operator

Parameters

col	is the number of elements to decrement over, can be negative

Returns

a new iterator referring to the decremented value

7.69.3.7 template<typename ValueType = double, typename WorkingPrecision = long double> int desalvo_standard_library::table< ValueType, WorkingPrecision >::row_const_iterator::operator- (const row_const_iterator & *p2*) const [inline]

Take the difference between two iterators of the same type, ∗this-p2

Parameters

p2	is the rhs of ∗this-p2

Returns

the number of elements that must be transversed in order to get from ∗this to p2; can be negative.

7.69.3.8 template<typename ValueType = double, typename WorkingPrecision = long double> row_const_iterator& desalvo_standard_library::table< ValueType, WorkingPrecision >::row_const_iterator::operator-- ( ) [inline]

Standard prefix – operator

Returns

reference to self after the increment.

7.69.3.9 template<typename ValueType = double, typename WorkingPrecision = long double> row_const_iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::row_const_iterator::operator-- (int) [inline]

Postfix – operator

7.69.3.10 template<typename ValueType = double, typename WorkingPrecision = long double> row_const_iterator& desalvo_standard_library::table< ValueType, WorkingPrecision >::row_const_iterator::operator-= (int *col*) [inline]

Decrement equals operator

Parameters

col	is the number of elements to increment over, can be negative

Returns

reference to the iterator for chaining

7.69.3.11 template<typename ValueType = double, typename WorkingPrecision = long double> ValueType& desalvo_standard_library::table< ValueType, WorkingPrecision >::row_const_iterator::operator-> ( ) const [inline]

Dereference operator

Returns

the value the const_iterator points to. Equivalent to operator* by dereferencing twice.

7.69.3.12 template<typename ValueType = double, typename WorkingPrecision = long double> ValueType& desalvo_standard_library::table< ValueType, WorkingPrecision >::row_const_iterator::operator[] (int *n*) const [inline]

Random access operator, makes iterator feel more like a pointer

Parameters

n	is the offset, can be negative

Returns

a reference to the element referred to by the iterator and offset

7.69.4 Friends And Related Function Documentation

7.69.4.1 template<typename ValueType = double, typename WorkingPrecision = long double> bool operator< (const table::row_const_iterator & *lhs*, const table::row_const_iterator & *rhs*) [friend]

Tests for const_iterators in same row but strictly smaller column

Parameters

t	is the other const_iterator

Returns

true if const_iterators are equivalent

7.69.4.2 template<typename ValueType = double, typename WorkingPrecision = long double> bool operator== (const table::row_const_iterator & *lhs*, const table::row_const_iterator & *rhs*) [friend]

Tests for const_iterators in two equivalent positions

Parameters

t	is the other const_iterator

Returns

true if const_iterators are equivalent

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

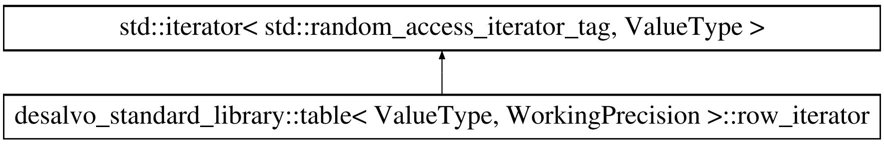
- DeSalvo Standard Library/desalvo/table.h

7.70 desalvo_standard_library::table< ValueType, WorkingPrecision >::row_iterator Class Reference

Random Access Iterator for Rows, muentry.

`#include <table.h>`

Inheritance diagram for desalvo_standard_library::table< ValueType, WorkingPrecision >::row_iterator:



Public Member Functions

- row_iterator (table ∗initial_mat, int initial_row=0, int initial_col=0)
- row_iterator (const row_iterator &other)
- void swap (row_iterator &other)
- row_iterator & operator= (row_iterator to_copy)
- row_iterator & operator++ ()
- row_iterator operator++ (int)
- row_iterator & operator+= (int col)
- row_iterator operator+ (int col)
- row_iterator & operator-- ()
- row_iterator operator-- (int)
- row_iterator & operator-= (int col)
- row_iterator operator- (int col)
- int operator- (const row_iterator &p2) const
- ValueType & operator∗ () const
- ValueType & operator-> () const
- ValueType & operator[] (int n)

Friends

- bool operator== (const table::row_iterator &lhs, const table::row_iterator &rhs)
- bool operator< (const table::row_iterator &lhs, const table::row_iterator &rhs)

7.70.1 Detailed Description

template<typename ValueType = double, typename WorkingPrecision = long double>class desalvo_standard_library::table< ValueType, WorkingPrecision >::row_iterator

Random Access Iterator for Rows, muentry.

This class is designed to be a RANDOM ACCESS ITERATOR for a given row of the entry.

The row is modifiable so that it can change which row it is pointing to. The column is modified along with the pointer so that it is easier to keep track of bounds.

```
#include <iostream>
#include <vector>
#include "std_cout.h"
#include "desalvo/table.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Make a vector of values
std::vector<int> v(100);

// initialize values to 0,1,2,...,99
std::iota(std::begin(v), std::end(v), 0);
```

```
// Initialize 10x10 table using 10 numbers for each row from v
dsl::table<int> t(10,10,std::begin(v));

// print out the table of values first
std::cout << "t: \n" << t << std::endl << std::endl;

// Iterate through every other row, squaring each element in each row
for(auto i = 0; i<10; i += 2) {
// Square each value
std::for_each(t.begin_row(i), t.end_row(i), [](int& a) { a *= a; });
}

std::cout << "Printing every third column ... \n";
// Iterate through every third column, print it out
for(auto i = 0; i<10; i += 3) {
// Print every other row
std::cout << "{";
std::for_each(t.cbegin_column(i), t.cend_column(i), [](int a) { std::cout << a<<","; });
std::cout << "}\n\n";
}

// Sort each row...no good reason, just want to demonstrate that the iterators work even for algorithms
       which require random access iterators.  Note that begin_row and end_row return objects, not raw pointers.  Use
       begin_row_raw and end_row_raw to obtain the raw pointer types.
for(auto i = 0;i<10;++i)
std::sort(t.begin_row(i), t.end_row(i));

std::cout << "After all that, sort elements in each row, and print t again:\n";
std::cout << t << std::endl;

return 0;
}
```

Should produce output

```
t:
{{0,1,2,3,4,5,6,7,8,9},
{10,11,12,13,14,15,16,17,18,19},
{20,21,22,23,24,25,26,27,28,29},
{30,31,32,33,34,35,36,37,38,39},
{40,41,42,43,44,45,46,47,48,49},
{50,51,52,53,54,55,56,57,58,59},
{60,61,62,63,64,65,66,67,68,69},
{70,71,72,73,74,75,76,77,78,79},
{80,81,82,83,84,85,86,87,88,89},
{90,91,92,93,94,95,96,97,98,99}}

Printing every third column ...
{0,10,400,30,1600,50,3600,70,6400,90,}

{9,13,529,33,1849,53,3969,73,6889,93,}

{36,16,676,36,2116,56,4356,76,7396,96,}

{81,19,841,39,2401,59,4761,79,7921,99,}

After all that, sort elements in each row, and print t again:
{{0,1,4,9,16,25,36,49,64,81},
{10,11,12,13,14,15,16,17,18,19},
{400,441,484,529,576,625,676,729,784,841},
{30,31,32,33,34,35,36,37,38,39},
{1600,1681,1764,1849,1936,2025,2116,2209,2304,2401},
{50,51,52,53,54,55,56,57,58,59},
{3600,3721,3844,3969,4096,4225,4356,4489,4624,4761},
{70,71,72,73,74,75,76,77,78,79},
{6400,6561,6724,6889,7056,7225,7396,7569,7744,7921},
{90,91,92,93,94,95,96,97,98,99}}
```

7.70.2 Constructor & Destructor Documentation

7.70.2.1 template<typename ValueType = double, typename WorkingPrecision = long double> desalvo_standard_library::table< ValueType, WorkingPrecision >::row_iterator::row_iterator (table * *initial_mat,* int *initial_row* = 0, int *initial_col* = 0) [inline]

Construct by table object

Parameters

T	is the table object with data
r	is the row number.
col	is the column

7.70.2.2 template<typename ValueType = double, typename WorkingPrecision = long double> desalvo_standard_library::table< ValueType, WorkingPrecision >::row_iterator::row_iterator (const row_iterator & *other*) [inline]

Copy constructor, same as default

Parameters

other	is the iterator from which to copy from

7.70.3 Member Function Documentation

7.70.3.1 template<typename ValueType = double, typename WorkingPrecision = long double> ValueType& desalvo_standard_library::table< ValueType, WorkingPrecision >::row_iterator::operator* ( ) const [inline]

Dereference operator

Returns

the value the iterator points to.

7.70.3.2 template<typename ValueType = double, typename WorkingPrecision = long double> row_iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::row_iterator::operator+ (int *col*) [inline]

Increment operator

Parameters

col	is the number of elements to increment over, can be negative

Returns

a new iterator referring to the incremented value

7.70.3.3 template<typename ValueType = double, typename WorkingPrecision = long double> row_iterator& desalvo_standard_library::table< ValueType, WorkingPrecision >::row_iterator::operator++ ( ) [inline]

Standard prefix ++ operator

Returns

reference to self after the increment.

7.70.3.4 template<typename ValueType = double, typename WorkingPrecision = long double> row_iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::row_iterator::operator++ (int) [inline]

Postfix ++ operator

7.70.3.5 template<typename ValueType = double, typename WorkingPrecision = long double> row_iterator& desalvo_standard_library::table< ValueType, WorkingPrecision >::row_iterator::operator+= (int *col*) [inline]

Increment equals operator

Parameters

col	is the number of elements to increment over, can be negative

Returns

reference to the iterator for chaining

7.70.3.6 template<typename ValueType = double, typename WorkingPrecision = long double> row_iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::row_iterator::operator- (int *col*) [inline]

Decrement operator

Parameters

col	is the number of elements to decrement over, can be negative

Returns

a new iterator referring to the decremented value

7.70.3.7 template<typename ValueType = double, typename WorkingPrecision = long double> int desalvo_standard_library::table< ValueType, WorkingPrecision >::row_iterator::operator- (const row_iterator & *p2*) const [inline]

Take the difference between two iterators of the same type, ∗this-p2

Parameters

p2	is the rhs of ∗this-p2

Returns

the number of elements that must be transversed in order to get from ∗this to p2; can be negative.

7.70.3.8 template<typename ValueType = double, typename WorkingPrecision = long double> row_iterator& desalvo_standard_library::table< ValueType, WorkingPrecision >::row_iterator::operator-- ( ) [inline]

Standard prefix – operator

Returns

reference to self after the increment.

7.70.3.9 template<typename ValueType = double, typename WorkingPrecision = long double> row_iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::row_iterator::operator-- (int) [inline]

Postfix – operator

7.70.3.10 template<typename ValueType = double, typename WorkingPrecision = long double> row_iterator& desalvo_standard_library::table< ValueType, WorkingPrecision >::row_iterator::operator-= (int *col*) [inline]

Decrement equals operator

Parameters

col	is the number of elements to increment over, can be negative

Returns

reference to the iterator for chaining

7.70.3.11 template<typename ValueType = double, typename WorkingPrecision = long double> ValueType& desalvo_standard_library::table< ValueType, WorkingPrecision >::row_iterator::operator-> ( ) const [inline]

Dereference operator

Returns

the value the iterator points to. Equivalent to operator∗ by dereferencing twice.

7.70.3.12 template<typename ValueType = double, typename WorkingPrecision = long double> row_iterator& desalvo_standard_library::table< ValueType, WorkingPrecision >::row_iterator::operator= (row_iterator *to_copy*) [inline]

Assignment operator, follows the Copy & Swap idiom

Parameters

to_copy	is the iterator from which to copy from

Returns

a reference to the iterator, for chaining.

7.70.3.13 template<typename ValueType = double, typename WorkingPrecision = long double> ValueType& desalvo_standard_library::table< ValueType, WorkingPrecision >::row_iterator::operator[] (int *n*) [inline]

Random access operator, makes the iterator act like a pointer.

Parameters

n	is the offset, can be negative.

Returns

reference to the element referred to by the iterator and offset.

7.70.3.14 template< typename ValueType = double, typename WorkingPrecision = long double> void desalvo_standard_library::table< ValueType, WorkingPrecision >::row_iterator::swap (row_iterator & *other*) [inline]

Standard swap between two row_iterator s, even those referring to different tables

Parameters

other	is another row_iterator

7.70.4 Friends And Related Function Documentation

7.70.4.1 template< typename ValueType = double, typename WorkingPrecision = long double> bool operator< (const table::row_iterator & *lhs*, const table::row_iterator & *rhs*) [friend]

Tests for iterators in same row but strictly smaller column

Parameters

t	is the other iterator

Returns

true if iterators are equivalent

7.70.4.2 template< typename ValueType = double, typename WorkingPrecision = long double> bool operator== (const table::row_iterator & *lhs*, const table::row_iterator & *rhs*) [friend]

Tests for iterators in two equivalent positions

Parameters

t	is the other iterator

Returns

true if iterators are equivalent

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

- DeSalvo Standard Library/desalvo/table.h

7.71 desalvo_standard_library::sequence_parameters< T1, Args > Class Template Reference

Public Member Functions

- **sequence_parameters** (T1 t, Args...args)
- void **replace_with** (std::tuple< T1, Args...> &&params)
- void **replace_with** (const std::tuple< T1, Args...> &params)
- template<typename T >
  T **get** (const size_t index)

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

- DeSalvo Standard Library/desalvo/sampling.h

7.72 desalvo_standard_library::set_partition< KeyType, ValueType, WorkingPrecision > Class Template Reference

Classes

- class generator
- class object

Public Member Functions

- template<typename Parameters = std::vector<int>, typename URNG = std::mt19937_64>
  object< Parameters > **random** (ValueType n, URNG &g=generator_64)

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

- DeSalvo Standard Library/desalvo/set_partition.h

7.73 desalvo_standard_library::shrinking_set< T, Comparison > Class Template Reference

initialized with a set of objects, then efficiently erases and resets again keeping non-erased elements in sorted order

```
#include <shrinking_set.h>
```

Public Member Functions

- shrinking_set ()
- shrinking_set (std::initializer_list< T > &&list)
- template<typename U >
  shrinking_set (U &&list)
- template<typename InputIterator >
  shrinking_set (InputIterator it, InputIterator it_stop)
- void reinitialize (const std::vector< T > &list)
- void reinitialize (std::vector< T > &&list)
- template<typename U >
  void reinitialize (U &&list)
- template<typename InputIterator >
  void reinitialize (InputIterator it, InputIterator it_stop)
- void reinitialize_with_ordered (const std::vector< T > &list)

- void reinitialize_with_ordered (std::vector< T > &&list)
- template<typename U >
  void reinitialize_with_ordered (U &&list)
- template<typename U >
  std::vector< T >::iterator find (U &&t) const
- template<typename U >
  bool erase (U &&t)
- template<typename UnaryPredicate >
  void remove_if (UnaryPredicate pred)
- void unerase ()
- T & operator[] (size_t index)
- void reset ()
- size_t size () const
- bool empty () const
- std::vector< T >::iterator begin () const
- std::vector< T >::iterator end () const
- std::vector< T >::const_iterator cbegin () const
- std::vector< T >::const_iterator cend () const

Friends

- std::ostream & operator<< (std::ostream &out, const shrinking_set< T, Comparison > &s)

7.73.1 Detailed Description

template<typename T, typename Comparison = std::less<T>>class desalvo_standard_library::shrinking_set< T, Comparison >

initialized with a set of objects, then efficiently erases and resets again keeping non-erased elements in sorted order

Template Parameters

T	is the underlying type of objects

The motivation for this class is from random sampling of Sudoku matrices and Latin squares, where one starts with a set of objects {1,2,...,n} and then erases elements one at a time according to some set of constraints.

If you just use std::set, then there are a lot of memory management costs. Instead, the idea is to always store the entire set of objects, and then rotate elements at the end to maintain sorted order. For example, suppose we start with {1,2,...,9} and then eliminate the 5, then 7, then 8 by rotating and updating a pointer. The set would store its elements in a vector with the following order

{1,2,3,4,5,6,7,8,9}

{1,2,3,4,6,7,8,9,5}

{1,2,3,4,6,8,9,7,5}

{1,2,3,4,6,9,8,7,5}

The size of the list and valid set of elements is kept track of by a pointer which points to one after the last element in the allowable list parts. It starts out pointing to one after the 9, then at the 5, then at the 7, then at the 8. To the user of the class, it will appear as though the list consists of the set of elements

{1,2,3,4,5,6,7,8,9}

{1,2,3,4,6,7,8,9}

{1,2,3,4,6,8,9}

{1,2,3,4,6,9}

When reset is called, the pointer returns to one after the last position, or equivalent to std::end(...), and the initial order is restored, and so when reset is called at this point the user of the class now has

{1,2,3,4,5,6,7,8,9}

which contains all elements in the specified order.

```
ddsl::shrinking_set_unordered<short> row({1,2,3,4,5,6,7,8,9});

cout << row << endl;

row.erase(3);
row.erase(6);
row.erase(6);
row.erase(8);
row.erase(7);
row.erase(1);
row.erase(2);
row.erase(3);
row.erase(4);
row.erase(5);
row.erase(6);
row.erase(7);
row.erase(8);
row.erase(5);
row.erase(6);
row.erase(7);
row.erase(8);
cout << row << endl;

row.reset(true);
row.erase(9);
row.erase(8);

row.erase(7);
row.erase(8);

cout << row << endl;


std::vector<short> v {1,2,3,4,5,6,7,8,9};

//ddsl::shrinking_set_unordered<short> row(std::begin(v), std::end(v));
ddsl::shrinking_set<short> row(std::begin(v), std::end(v));

cout << row << endl;

row.erase(3);
row.erase(6);
row.erase(6);
row.erase(8);
row.erase(1);
row.erase(4);
row.erase(2);
cout << row << endl;

row.reset();
row.erase(9);
row.erase(1);
row.erase(8);
row.erase(3);

//row.erase(7);
//row.erase(10);

cout << row << endl;

row.unerase();
cout << row << endl;

auto x = row.find(1);

if(x != std::end(row))
cout << *x << std::endl;
else
cout << "element not found " << std::endl;

std::vector<int> v2 = dsl::range(100);

auto x2 = dsl::binary_search_iterator(std::begin(v2), std::end(v2), 27);

std::cout << *x2 << std::endl;

row.remove_if([](int a) { return a <=5;});

std::cout << row << std::endl;
```

7.73.2 Constructor & Destructor Documentation

7.73.2.1 template<typename T, typename Comparison = std::less<T>> desalvo_standard_library::shrinking_set< T, Comparison >::shrinking_set ( ) [inline]

Constructs empty collection

```
#include <initializer_list>
#include "desalvo/shrinking_set.h"
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Create default set of size 0;
dsl::shrinking_set<char> v;

dsl::print(v,"\n");

return 0;
}
```

Should produce output

```
{}
```

7.73.2.2 template<typename T, typename Comparison = std::less<T>> desalvo_standard_library::shrinking_set< T, Comparison >::shrinking_set (std::initializer_list< T > && *list*) [inline]

Constructs collection of objects using an initializer list of values in any order

Parameters

list	is the collection of objects
	`#include "desalvo/shrinking_set.h"`
	`#include "desalvo/std_cout.h"`
	`namespace dsl = desalvo_standard_library;`
	`int main(int argc, const char * argv[]) {`
	`// Initialize set with letters a,b,c,d,e,f in any order`
	`dsl::shrinking_set<char> v3 {'f','e','c','d','a','b'};`
	`dsl::print(v3,"\n");`
	`return 0;`
	`}`
	Should produce output
	`{a,b,c,d,e,f}`

7.73.2.3 template<typename T, typename Comparison = std::less<T>> template<typename U > desalvo_standard_library::shrinking_set< T, Comparison >::shrinking_set (U && *list*) [inline]

Constructs collection of objects using any object which can be copy constructed by an std::vector

Template Parameters

U	is the type of the collection of objects

Parameters

list	is the collection of objects

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// abacadaba
auto letters {'a','b','a','c','a','d','a','b','a'};

dsl::shrinking_set<char> v(letters);
dsl::print(v,"\n");

std::list<char> list_of_chars {'z','y','x','w'};

dsl::shrinking_set<char> v2(list_of_chars);
dsl::print(v2,"\n");

std::set<char> set_of_chars {'l','m','a','o'};
dsl::shrinking_set<char> v3(set_of_chars);
dsl::print(v3,"\n");

std::array<int,5> array_of_ints {3,1,4,1,5};
dsl::shrinking_set<int> v4(array_of_ints);
dsl::print(v4,"\n");

double p[3] {3.14159265, 2.718281828, 2};
dsl::shrinking_set<double> v5(p);
dsl::print(v5,"\n");

return 0;
}
```

Should produce output

```
{a,b,c,d}
{w,x,y,z}
{a,l,m,o}
{1,3,4,5}
{2,2.71828,3.14159}
```

7.73.2.4 template< typename T, typename Comparison = std::less< T > > template< typename InputIterator > desalvo_standard_library::shrinking_set< T, Comparison >::shrinking_set (InputIterator *it*, InputIterator *it_stop*) [inline]

Constructs collection of objects using a range of values specified by input iterators

Template Parameters

InputIterator	is any input iterator type

Parameters

it	is an iterator referring to the first element
it_stop	is an iterator referring to one after the last element

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// abacadaba
auto letters {'a','b','a','c','a','d','a','b','a'};

dsl::shrinking_set<char> v(std::begin(letters), std::end(letters));
dsl::print(v,"\n");

std::list<char> list_of_chars {'z','y','x','w'};

dsl::shrinking_set<char> v2(std::begin(list_of_chars),std::end(list_of_chars));
dsl::print(v2,"\n");

std::set<char> set_of_chars {'l','m','a','o'};
dsl::shrinking_set<char> v3(std::begin(set_of_chars),std::end(set_of_chars));
dsl::print(v3,"\n");

std::array<int,5> array_of_ints {3,1,4,1,5}; // random access iterator
dsl::shrinking_set<int> v4(std::begin(array_of_ints),std::end(array_of_ints)-2);
dsl::print(v4,"\n");

double p[3] {3.14159265, 2.718281828, 2}; // random access iterator
dsl::shrinking_set<double> v5(std::begin(p), std::end(p)-1);
dsl::print(v5,"\n");

return 0;
}
```

Should produce output

```
{a,b,c,d}
{w,x,y,z}
{a,l,m,o}
{1,3,4}
{2.71828,3.14159}
```

7.73.3 Member Function Documentation

7.73.3.1 template<typename T, typename Comparison = std::less<T>> std::vector<T>::iterator desalvo_standard_library::shrinking_set< T, Comparison >::begin ( ) const [inline]

returns iterator to the first element of the collection

Returns

iterator to the first element of the collection

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

dsl::shrinking_set<int> v {3,2,1,4,5,6,7,8,0};
dsl::print(v,"\n");

std::vector<int> v2;

// copy all even elements to container v2
std::copy_if(std::begin(v), std::end(v), std::back_inserter(v2),[](int a) { return a%2==0;});

dsl::print(v2,"\n");

return 0;
}
```

Should produce output

```
{0,1,2,3,4,5,6,7,8}
{0,2,4,6,8}
```

7.73.3.2 template<typename T, typename Comparison = std::less<T>> std::vector<T>::const_iterator desalvo_standard_library::shrinking_set< T, Comparison >::cbegin ( ) const [inline]

returns iterator to the first element of the collection

Returns

iterator to the first element of the collection

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

dsl::shrinking_set<int> v {3,2,1,4,5,6,7,8,0};
dsl::print(v,"\n");

// Print out the squares of each value

// C++14 syntax
//std::for_each(std::cbegin(v), std::cend(v), [](int a) { return std::cout << a*a << ",";});

std::for_each(v.cbegin(), v.cend(), [&](int a) { std::cout << a*a << ",";});

return 0;
}
```

Should produce output

```
{0,1,2,3,4,5,6,7,8}
0,1,4,9,16,25,36,49,64,
```

7.73.3.3 template<typename T, typename Comparison = std::less<T>> std::vector<T>::const_iterator desalvo_standard_library::shrinking_set< T, Comparison >::cend ( ) const [inline]

returns iterator to the one after the last element of the collection

Returns

iterator to the last element of the collection

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

dsl::shrinking_set<int> v {3,2,1,4,5,6,7,8,0};
dsl::print(v,"\n");

// Print out the squares of each value

// C++14 syntax
//std::for_each(std::cbegin(v), std::cend(v), [](int a) { return std::cout << a*a << ",";});

std::for_each(v.cbegin(), v.cend(), [&](int a) { std::cout << a*a << ",";});

return 0;
}
```

Should produce output

```
{0,1,2,3,4,5,6,7,8}
0,1,4,9,16,25,36,49,64,
```

7.73.3.4 template<typename T, typename Comparison = std::less<T>> bool desalvo_standard_library::shrinking_set< T, Comparison >::empty ( ) const [inline]

Check whether or not the container is empty

Returns

true if the container is empty

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

dsl::shrinking_set<int> v {3,2,1,4,5,6,7,8,0};
dsl::print(v,"\n");

std::cout << "Now remove all multiples of 2, 3, or 5 \n";
v.remove_if([](int a) { return a%3==0;});
v.remove_if([](int a) { return a%2==0;});
v.remove_if([](int a) { return a%5==0;});

std::cout << "After removing all multiples of 2, 3, or 5\n";
std::cout << "is v empty?  "<< (v.empty()? "yes" : "no" ) << std::endl;

if(!v.empty()) dsl::print(v,"\n");

return 0;
}
```

Should produce output

```
{0,1,2,3,4,5,6,7,8}
Now remove all multiples of 2, 3, or 5
After removing all multiples of 2, 3, or 5
is v empty?  no
{1,7}
```

7.73.3.5 template<typename T, typename Comparison = std::less<T>> std::vector<T>::iterator desalvo_standard_library::shrinking_set< T, Comparison >::end ( ) const [inline]

returns iterator to the one after the last element of the collection

Returns

iterator to the last element of the collection

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

dsl::shrinking_set<int> v {3,2,1,4,5,6,7,8,0};
dsl::print(v,"\n");

std::vector<int> v2;

// copy all even elements to container v2
std::copy_if(std::begin(v), std::end(v), std::back_inserter(v2),[](int a) { return a%2==0;});

dsl::print(v2,"\n");

return 0;
}
```

Should produce output

```
{0,1,2,3,4,5,6,7,8}
{0,2,4,6,8}
```

7.73.3.6 template<typename T, typename Comparison = std::less<T>> template<typename U > bool desalvo_standard_library::shrinking_set< T, Comparison >::erase (U && *t*) [inline]

Erases an element from the list

Template Parameters

U	is any type which can be cast to type T

Parameters

t	is the value of an element to erase

Returns

true if element was in the list and erased, false otherwise

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

dsl::shrinking_set<int> v {3,1,4,1,5,9,2,6,5,3,5,8};
dsl::print(v,"\n");

v.erase(3);
v.erase(6);
v.erase(9);

dsl::print(v,"\n");

return 0;
}
```

Should produce output

```
{1,2,3,4,5,6,8,9}
{1,2,4,5,8}
```

7.73.3.7 template<typename T, typename Comparison = std::less<T>> template<typename U > std::vector<T>::iterator desalvo_standard_library::shrinking_set< T, Comparison >::find (U && *t*) const [inline]

Find a particular element inside of the container.

Template Parameters

U	is any type which can be cast to type T

Parameters

t	is the value of an element to search for

Returns

iterator to location, or to end of the container

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// abacadaba
auto letters {'a','b','a','c','a','d','a','b','a'};
```

```
dsl::shrinking_set<char> v(std::begin(letters), std::end(letters));
dsl::print(v,"\n");

// Return iterator to character c
auto it = v.find('c');

// print all letters from 'c' until the end.
std::for_each(it, std::end(v), [](char c) { std::cout <<c<<",";});

return 0;
}
```

Should produce output

```
{a,b,c,d}
c,d,
```

7.73.3.8 template<typename T, typename Comparison = std::less<T>> T& desalvo_standard_library::shrinking_set< T, Comparison >::operator[] (size_t *index*) [inline]

Random access operator

Parameters

index	is the element index

Returns

the value at the given index

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

dsl::shrinking_set<int> v {3,1,4,1,5,9,2,6,5,3,5,8};
dsl::print(v,"\n");

std::cout << "First elements: " << v[0] << std::endl;

std::cout << "Sum of first three elements: " << v[0]+v[1]+v[2] << std::endl;

return 0;
}
```

Should produce output

```
{1,2,3,4,5,6,8,9}
First elements: 1
Sum of first three elements: 6
```

7.73.3.9 template<typename T, typename Comparison = std::less<T>> void desalvo_standard_library::shrinking_set< T, Comparison >::reinitialize (const std::vector< T > & *list*) [inline]

reinitialize using a vector of the same type

Parameters

list	is another collection of objects

```
#include <numeric>
#include "desalvo/shrinking_set.h"
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Create vector with entries {1,2,...,9}
std::vector<int> v(9);
std::iota(std::begin(v), std::end(v), 1);

// Initialize entries using iterators
dsl::shrinking_set<int> v2(std::begin(v), std::end(v));

dsl::print(v2, "\n");

// Create new vector of values {-10,-9,...,9,10}
std::vector<int> v3(21);
std::iota(std::begin(v3), std::end(v3), -10);

// Reinitialize shrinking set with those values
v2.reinitialize(v3);

dsl::print(v2, "\n");

return 0;
}
```

Should produce output

```
{1,2,3,4,5,6,7,8,9}
{-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10}
```

7.73.3.10 template<typename T, typename Comparison = std::less<T>> void desalvo_standard_library::shrinking_set< T, Comparison >::reinitialize (std::vector< T > && *list*) [inline]

reinitialize using an rvalue reference of the same type

Parameters

list	is an expiring collection of objects

```
#include <numeric>
#include "desalvo/shrinking_set.h"
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Create vector with entries {1,2,...,9}
std::vector<int> v(9);
std::iota(std::begin(v), std::end(v), 1);

// Initialize entries using iterators
dsl::shrinking_set<int> v2(std::begin(v), std::end(v));

dsl::print(v2, "\n");

// Reinitialize shrinking set with custom values
v2.reinitialize({1,2,3,4,5});

dsl::print(v2, "\n");

return 0;
}
```

Should produce output

```
{1,2,3,4,5,6,7,8,9}
{1,2,3,4,5}
```

7.73.3.11 template<typename T, typename Comparison = std::less<T>> template<typename U > void desalvo_standard_library::shrinking_set< T, Comparison >::reinitialize (U && *list*) [inline]

reinitialize using another collection of objects which supplies at least an input iterator

Template Parameters

U	is a collection of objects with an input iterator

Parameters

list	is the collection of objects

```
#include <numeric>
#include "desalvo/shrinking_set.h"
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Create vector with entries {1,2,...,9}
std::vector<int> v(9);
std::iota(std::begin(v), std::end(v), 1);

// Initialize entries using iterators
dsl::shrinking_set<int> v2(std::begin(v), std::end(v));

dsl::print(v2, "\n");

// Reinitialize shrinking set with custom values
v2.reinitialize({4,5,3,2,1});

dsl::print(v2, "\n");

return 0;
}
```

Should produce output

```
{1,2,3,4,5,6,7,8,9}
{1,2,3,4,5}
```

7.73.3.12 template<typename T, typename Comparison = std::less<T>> template<typename InputIterator > void desalvo_standard_library::shrinking_set< T, Comparison >::reinitialize (InputIterator *it*, InputIterator *it_stop*) [inline]

reinitialize using another pair of input iterators

Template Parameters

InputIterator	is any input iterator type

Parameters

it	is an iterator referring to the first element
it_stop	is an iterator referring to one after the last element

```
#include "desalvo/shrinking_set.h"
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Create vector
std::vector<int> v {1,9,2,8,3,7,4,6,5,5,4,3,2,1};

// Initialize entries using iterators
dsl::shrinking_set<int> v2(std::begin(v), std::end(v));

dsl::print(v2, "\n");

// Create new vector of values {-10,-9,...,9,10}
std::vector<int> v3 {-10,10,-9,9,-8,8,-7,7,-6,6,-5,5,-4,4};

// Reinitialize shrinking set with those values
v2.reinitialize(std::begin(v3), std::end(v3));

dsl::print(v2, "\n");

return 0;
}
```

Should produce output

```
{1,2,3,4,5,6,7,8,9}
{-10,-9,-8,-7,-6,-5,-4,4,5,6,7,8,9,10}
```

7.73.3.13 template<typename T, typename Comparison = std::less<T>> void desalvo_standard_library::shrinking_set< T, Comparison >::reinitialize_with_ordered (const std::vector< T > & *list*) [inline]

reinitializes without sorting, specialized for std::vector since it will be slightly faster than in general

Parameters

list	must be sorted

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

std::vector<char> ordered_name() { return std::vector<char>{'a','l'};};

int main(int argc, const char * argv[]) {

// abacadaba
auto letters {'a','b','a','c','a','d','a','b','a'};

dsl::shrinking_set<char> v(std::begin(letters), std::end(letters));
dsl::print(v,"\n");

std::vector<char> vector_of_chars {'g','p','s'};
v.reinitialize_with_ordered(vector_of_chars);
dsl::print(v,"\n");

v.reinitialize_with_ordered(ordered_name());
dsl::print(v,"\n");

return 0;
}
```

Should produce output

```
{a,b,c,d}
{g,p,s}
{a,l}
```

7.73.3.14 template<typename T, typename Comparison = std::less<T>> void desalvo_standard_library::shrinking_set< T, Comparison >::reinitialize_with_ordered (std::vector< T > && *list*) [inline]

reinitialize using sorted expiring collection, specialized for std::vector since it will be slightly faster than in general

Parameters

list	is an expiring, sorted collection

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

std::vector<char> ordered_name() { return std::vector<char>{'a','l'};};

int main(int argc, const char * argv[]) {

// abacadaba
auto letters {'a','b','a','c','a','d','a','b','a'};

dsl::shrinking_set<char> v(std::begin(letters), std::end(letters));
dsl::print(v,"\n");

v.reinitialize_with_ordered(ordered_name());
dsl::print(v,"\n");

return 0;
}
```

Should produce output

```
{a,b,c,d}
{a,l}
```

7.73.3.15 template<typename T, typename Comparison = std::less<T>> template<typename U > void desalvo_standard_library::shrinking_set< T, Comparison >::reinitialize_with_ordered (U && *list*) [inline]

reinitialize using another collection of objects which supplies at least an input iterator, the input list MUST ALREADY BE SORTED.

Template Parameters

U	is a collection of objects with an input iterator

Parameters

list	is the collection of objects
	`#include <numeric>`
	`#include "desalvo/shrinking_set.h"`
	`#include "desalvo/std_cout.h"`
	
	`namespace dsl = desalvo_standard_library;`
	
	`int main(int argc, const char * argv[]) {`
	
	`// Create vector with entries {1,2,...,9}`
	`std::vector<int> v(9);`
	`std::iota(std::begin(v), std::end(v), 1);`
	
	`// Initialize entries using iterators`
	`dsl::shrinking_set<int> v2(std::begin(v), std::end(v));`
	
	`dsl::print(v2, "\n");`
	
	`// Reinitialize shrinking set with custom values`
	`v2.reinitialize({1,2,3,4,5});`
	
	`dsl::print(v2, "\n");`
	
	`return 0;`
	`}`
	
	Should produce output
	
	`{1,2,3,4,5,6,7,8,9}`
	`{1,2,3,4,5}`

Example 2:

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// abacadaba
auto letters {'a','b','a','c','a','d','a','b','a'};

dsl::shrinking_set<char> v(std::begin(letters), std::end(letters));
dsl::print(v,"\n");

std::vector<char> vector_of_chars {'s','t','u'};
v.reinitialize_with_ordered(vector_of_chars);
dsl::print(v,"\n");

std::list<char> list_of_chars {'w','x','y','z'};

v.reinitialize_with_ordered(list_of_chars);
dsl::print(v,"\n");

std::set<char> set_of_chars {'a','b','c','d'};
v.reinitialize_with_ordered(set_of_chars);
dsl::print(v,"\n");

return 0;
}
```

Should produce output

```
{a,b,c,d}
{s,t,u}
{w,x,y,z}
{a,b,c,d}
```

7.73.3.16 template< typename T, typename Comparison = std::less< T > > template< typename UnaryPredicate > void desalvo_standard_library::shrinking_set< T, Comparison >::remove_if (UnaryPredicate *pred*) [inline]

Removes all elements which return true when input into the unary predicate function

Template Parameters

UnaryPredicate	is any function object type

Parameters

pred	is an object with a operator(T)->bool function defined

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

dsl::shrinking_set<int> v {3,1,4,1,5,9,2,6,5,3,5,8};
dsl::print(v,"\n");

// Instead of these lines ...
//v.erase(3);
//v.erase(6);
//v.erase(9);

v.remove_if( [](int a) { return a%3 == 0;});

dsl::print(v,"\n");

return 0;
}
```

Should produce output

```
{1,2,3,4,5,6,8,9}
{1,2,4,5,8}
```

7.73.3.17 template<typename T, typename Comparison = std::less<T>> void desalvo_standard_library::shrinking_set< T, Comparison >::reset ( ) [inline]

Resets the set to have all elements again

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Three little ducks went out one day ...
dsl::shrinking_set<int> v {3,2,1};
dsl::print(v,"\n");

// over the bridge and far away ...
v.erase(3);

// Mother duck said, "Quack quack quack"
// Two little ducks came back
dsl::print(v,"\n");

// Two little ducks went out one day, over the bridge and far away ...
v.erase(1);

// Mother duck said, "Quack quack quack"
// One little duck came back
dsl::print(v,"\n");

// One little duck went out one day, over the bridge and far away ...
v.erase(2);

// Mother duck said, "Quack quack quack"
// No little ducks came back
dsl::print(v,"\n");

// Sad mother duck went out one day, over the bridge and far away ...
// Mother duck said, "Quack quack quack"
v.reset();

// All the little ducks came back!
```

```
dsl::print(v,"\n");

return 0;
}
```

Should produce output

```
{1,2,3}
{1,2}
{2}
{}
{1,2,3}
```

7.73.3.18 template<typename T, typename Comparison = std::less<T>> size_t desalvo_standard_library::shrinking_set< T, Comparison >::size ( ) const [inline]

returns the size of the set

Returns

the size of the set

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

dsl::shrinking_set<int> v {3,2,1,4,5,6,7,8,0};
dsl::print(v,"\n");

std::cout << "size of v: "<<v.size() << std::endl;;

v.remove_if([](int a) { return a%3==0;});

std::cout << "After removing all multiples of 3\n";
std::cout << "size of v: "<<v.size() << std::endl;

return 0;
}
```

Should produce output

```
{0,1,2,3,4,5,6,7,8}
size of v: 9
After removing all multiples of 3
size of v: 6
```

7.73.3.19 template<typename T, typename Comparison = std::less<T>> void desalvo_standard_library::shrinking_set< T, Comparison >::unerase ( ) [inline]

Unerases the element erased.

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

dsl::shrinking_set<int> v {3,1,4,1,5,9,2,6,5,3,5,8};
dsl::print(v,"\n");

// Instead of these lines ...
//v.erase(3);
//v.erase(6);
//v.erase(9);

v.remove_if( [](int a) { return a%3 == 0;});

dsl::print(v,"\n");
```

```
// Oops!  We wanted to keep the 9.
v.unerase();

dsl::print(v,"\n");

return 0;
}
```

Should produce output

```
{1,2,3,4,5,6,8,9}
{1,2,4,5,8}
{1,2,4,5,8,9}
```

7.73.4 Friends And Related Function Documentation

7.73.4.1 template<typename T, typename Comparison = std::less<T>> std::ostream& operator<< (std::ostream & *out*, const shrinking_set< T, Comparison > & *s*) [friend]

output operator, outputs of the form {#1,#2,...,#n}, where #1<#2<...<#n, specified by the Comparison template

Parameters

out	is the stream
s	is the collection of objects

Returns

the stream for chaining

```
#include <initializer_list>
#include "desalvo/shrinking_set.h"
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Create default set of size 0;
dsl::shrinking_set<char> v;

// Initialize set with letters a,b,c,d,e,f
// Cannot just use initializer list, since by default each character is a const char*
dsl::shrinking_set<char> v2(std::vector<char>({'f','e','c','d','a','b'}));

dsl::print(v,"\n");
dsl::print(v2);

return 0;
}
```

Should produce output

```
{}
{a,b,c,d,e,f}
```

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

- DeSalvo Standard Library/desalvo/shrinking_set.h

7.74 desalvo_standard_library::shrinking_set_unordered< T > Class Template Reference

initialized with a set of objects, then efficiently erases and resets again

`#include <shrinking_set.h>`

Public Member Functions

- shrinking_set_unordered ()
- shrinking_set_unordered (std::initializer_list< T > list)
- template<size_t N>
  shrinking_set_unordered (std::array< T, N > list)
- template<typename F >
  shrinking_set_unordered (F *list, size_t length)
- template<typename U >
  shrinking_set_unordered (U &&list)
- template<typename InputIterator >
  shrinking_set_unordered (InputIterator it, InputIterator it_stop)
- template<typename U >
  void reinitialize (U &&list)
- template<typename InputIterator >
  void reinitialize (InputIterator it, InputIterator it_stop)
- template<typename U >
  std::vector< T >::iterator find (U &&t)
- template<typename U >
  bool erase (U &&t)
- template<typename UnaryPredicate >
  void remove_if (UnaryPredicate pred)
- void unerase ()
- T & operator[] (size_t index)
- void reset (bool flag=false)
- size_t size ()
- bool empty ()
- std::vector< T >::iterator begin () const
- std::vector< T >::iterator end () const
- std::vector< T >::const_iterator cbegin () const
- std::vector< T >::const_iterator cend () const

Friends

- std::ostream & operator<< (std::ostream &out, const shrinking_set_unordered< T > &s)

7.74.1 Detailed Description

template<typename T>class desalvo_standard_library::shrinking_set_unordered< T >

initialized with a set of objects, then efficiently erases and resets again

Template Parameters

T	is the underlying type of objects

The motivation for this class is from random sampling of Sudoku matrices and Latin squares, where one starts with a set of objects {1,2,...,n} and then erases elements one at a time according to some set of constraints.

If you just use std::set, then there are a lot of memory management costs. Instead, the idea is to always store the entire set of objects, and then swap out elements at the end. For example, suppose we start with {1,2,...,9} and then eliminate the 5, then 7, then 8 by swapping and updating a pointer. The set would store its elements in a vector with the following order

{1,2,3,4,5,6,7,8,9}

{1,2,3,4,9,6,7,8,5}

{1,2,3,4,9,6,8,7,5}

{1,2,3,4,9,6,8,7,5}

The size of the list and valid set of elements is kept track of by a pointer which points to one after the last element in the allowable list parts. It starts out pointing to one after the 9, then at the 5, then at the 7, then at the 8. To the user of the class, it will appear as though the list consists of the set of elements

{1,2,3,4,5,6,7,8,9}

{1,2,3,4,9,6,7,8}

{1,2,3,4,9,6,8}

{1,2,3,4,9,6}

When reset is called, the pointer simply returns to one after the last position, or equivalent to std::end(...). The initial order is not respected, and so when reset is called at this point the user of the class now has

{1,2,3,4,9,6,8,7,5}

which contains all elements, but in some other order.

7.74.2 Constructor & Destructor Documentation

7.74.2.1 template< typename T > desalvo_standard_library::shrinking_set_unordered< T >::shrinking_set_unordered ( ) [inline]

Constructs empty collection

```
#include <initializer_list>
#include "desalvo/shrinking_set.h"
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Create default set of size 0;
dsl::shrinking_set_unordered<char> v;

dsl::print(v,"\n");

return 0;
}
```

Should produce output

```
{}
```

7.74.2.2 template< typename T> desalvo_standard_library::shrinking_set_unordered< T >::shrinking_set_unordered (std::initializer_list< T > *list*) [inline]

Constructs collection of objects using an initializer list of values in any order

Parameters

list	is the collection of objects

```
#include "desalvo/shrinking_set.h"
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Initialize set with letters a,b,c,d,e,f in any order
dsl::shrinking_set_unordered<char> v {'N','o','w',' ','i','s',' ','t','h'
    ,'e',' ','w','i','n','t','e','r',' ','o','f'};
dsl::print(v,"\n");

return 0;
}
```

Should produce output

```
{N,o,w, ,i,s,t,h,e,n,r,f}
```

7.74.2.3 template<typename T> template<size_t N> desalvo_standard_library::shrinking_set_unordered< T >::shrinking_set_unordered (std::array< T, N > *list*) [inline]

Constructs collection of objects using a collection stored in an object of type std::array<T,N>

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

std::array<int,5> array_of_ints {3,1,4,1,5};

dsl::shrinking_set_unordered<int> v4(array_of_ints);
dsl::print(v4,"\n");


return 0;
}
```

Should produce output

```
{3,1,4,5}
```

7.74.2.4 template<typename T> template<typename F > desalvo_standard_library::shrinking_set_unordered< T >::shrinking_set_unordered (F * *list*, size_t *length*) [inline]

Constructs collection of objects using a collection stored using pointers

Template Parameters

F	is a pointer type, with elements convertable to T

Parameters

list	is the pointer to the collection of elements
length	is the number of elements starting at list to consider

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

double p[3] {3.14159265, 2.718281828, 2};
dsl::shrinking_set_unordered<double> v5(p,3);
dsl::print(v5,"\n");


return 0;
}
```

Should produce output

```
{3.14159,2.71828,2}
```

7.74.2.5 template<typename T> template<typename U > desalvo_standard_library::shrinking_set_unordered< T >::shrinking_set_unordered (U && *list*) [inline]

Constructs collection of objects using any object which can be copy constructed by an std::vector

Template Parameters

U	is the type of the collection of objects

Parameters

list	is the collection of objects

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

std::list<char> list_of_chars {'z','y','x','w'};
dsl::shrinking_set_unordered<char> v2(list_of_chars);
dsl::print(v2,"\n");

std::set<char> set_of_chars {'r','o','f','l','m','a','o'};
dsl::shrinking_set_unordered<char> v3(set_of_chars);
dsl::print(v3,"\n");

std::array<int,5> array_of_ints {3,1,4,1,5};
dsl::shrinking_set_unordered<int> v4(array_of_ints);
dsl::print(v4,"\n");

return 0;
}
```

Should produce output

```
{z,y,x,w}
{a,f,l,m,o,r}
{3,1,4,5}
```

7.74.2.6 template<typename T> template<typename InputIterator > desalvo_standard_library::shrinking_set_unordered< T >::shrinking_set_unordered (InputIterator *it*, InputIterator *it_stop*) [inline]

Constructs collection of objects using a range of values specified by input iterators

Template Parameters

InputIterator	is any input iterator type

Parameters

it	is an iterator referring to the first element
it_stop	is an iterator referring to one after the last element

```
#include "desalvo/shrinking_set.h"
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

char p[] = {'N','o','w',' ','i','s',' ','t','h','e',' ','w','i','n','t','e','r',' ','o','f',' ','o','u','r
      ,' ','d','i','s','c','o','n','t','e','n','t'};

// Initialize set with letters a,b,c,d,e,f in any order
dsl::shrinking_set_unordered<char> v(std::begin(p), std::end(p));

dsl::print(v,"\n");

return 0;
}
```

Should produce output

```
{N,o,w, ,i,s,t,h,e,n,r,f,u,d,c}
```

Example 2:

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

std::list<char> list_of_chars {'z','y','x','w'};
dsl::shrinking_set_unordered<char> v2(std::begin(list_of_chars), std::end
      (list_of_chars));
dsl::print(v2,"\n");

std::set<char> set_of_chars {'r','o','f','l','m','a','o'};
dsl::shrinking_set_unordered<char> v3(std::begin(set_of_chars), std::end(
      set_of_chars));
dsl::print(v3,"\n");

std::array<int,5> array_of_ints {3,1,4,1,5};
dsl::shrinking_set_unordered<int> v4(std::begin(array_of_ints), std::end(
      array_of_ints));
dsl::print(v4,"\n");

return 0;
}
```

Should produce output

```
{z,y,x,w}
{a,f,l,m,o,r}
{3,1,4,5}
```

7.74.3 Member Function Documentation

7.74.3.1 template<typename T> std::vector<T>::iterator desalvo_standard_library::shrinking_set_unordered< T >::begin ( ) const [inline]

returns iterator to the first element of the collection

Returns

iterator to the first element of the collection

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

dsl::shrinking_set_unordered<int> v {3,2,1,4,5,6,7,8,0};
dsl::print(v,"\n");

std::vector<int> v2;

// copy all even elements to container v2
std::copy_if(std::begin(v), std::end(v), std::back_inserter(v2),[](int a) { return a%2==0;});

dsl::print(v2,"\n");

return 0;
}
```

Should produce output

```
{3,2,1,4,5,6,7,8,0}
{2,4,6,8,0}
```

7.74.3.2 template<typename T> std::vector<T>::const_iterator desalvo_standard_library::shrinking_set_-unordered< T >::cbegin ( ) const [inline]

returns iterator to the first element of the collection

Returns

iterator to the first element of the collection

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

dsl::shrinking_set_unordered<int> v {3,2,1,4,5,6,7,8,0};
dsl::print(v,"\n");

// Print out the squares of each value

// C++14 syntax
//std::for_each(std::cbegin(v), std::cend(v), [](int a) { return std::cout << a*a << ",";});

std::for_each(v.cbegin(), v.cend(), [&](int a) { std::cout << a*a << ",";});

return 0;
}
```

Should produce output

```
{3,2,1,4,5,6,7,8,0}
9,4,1,16,25,36,49,64,0,
```

7.74.3.3 template<typename T> std::vector<T>::const_iterator desalvo_standard_library::shrinking_set_-unordered< T >::cend ( ) const [inline]

returns iterator to the one after the last element of the collection

Returns

iterator to the last element of the collection

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

dsl::shrinking_set_unordered<int> v {3,2,1,4,5,6,7,8,0};
dsl::print(v,"\n");

// Print out the squares of each value

// C++14 syntax
//std::for_each(std::cbegin(v), std::cend(v), [](int a) { return std::cout << a*a << ",";});

std::for_each(v.cbegin(), v.cend(), [&](int a) { std::cout << a*a << ",";});

return 0;
}
```

Should produce output

```
{3,2,1,4,5,6,7,8,0}
9,4,1,16,25,36,49,64,0,
```

7.74.3.4 template<typename T> bool desalvo_standard_library::shrinking_set_unordered< T >::empty ( ) [inline]

Check whether or not the container is empty

Returns

true if the container is empty

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

dsl::shrinking_set_unordered<int> v {3,2,1,4,5,6,7,8,0};
dsl::print(v,"\n");

std::cout << "Now remove all multiples of 2, 3, or 5 \n";
v.remove_if([](int a) { return a%3==0;});
v.remove_if([](int a) { return a%2==0;});
v.remove_if([](int a) { return a%5==0;});

std::cout << "After removing all multiples of 2, 3, or 5\n";
std::cout << "is v empty?  "<< (v.empty()? "yes" : "no" ) << std::endl;

if(!v.empty()) dsl::print(v,"\n");

return 0;
}
```

Should produce output

```
{3,2,1,4,5,6,7,8,0}
Now remove all multiples of 2, 3, or 5
After removing all multiples of 2, 3, or 5
is v empty?  no
{7,1}
```

7.74.3.5 template<typename T> std::vector<T>::iterator desalvo_standard_library::shrinking_set_unordered< T >::end ( ) const [inline]

returns iterator to the one after the last element of the collection

Returns

iterator to the last element of the collection

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

dsl::shrinking_set_unordered<int> v {3,2,1,4,5,6,7,8,0};
dsl::print(v,"\n");

std::vector<int> v2;

// copy all even elements to container v2
std::copy_if(std::begin(v), std::end(v), std::back_inserter(v2),[](int a) { return a%2==0;});

dsl::print(v2,"\n");

return 0;
}
```

Should produce output

```
{3,2,1,4,5,6,7,8,0}
{2,4,6,8,0}
```

7.74.3.6 template<typename T> template<typename U > bool desalvo_standard_library::shrinking_set_unordered< T >::erase (U && *t*) [inline]

Erases an element from the list

Template Parameters

U	is any type which can be cast to type T

Parameters

t	is the value of an element to erase

Returns

true if element was in the list and erased, false otherwise

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

dsl::shrinking_set_unordered<int> v {3,1,4,1,5,9,2,6,5,3,5,8};
dsl::print(v,"\n");

v.erase(3);
v.erase(6);
v.erase(9);

dsl::print(v,"\n");

return 0;
}
```

Should produce output

```
{3,1,4,5,9,2,6,8}
{8,1,4,5,2}
```

7.74.3.7 template<typename T> template<typename U > std::vector<T>::iterator desalvo_standard_library::shrinking_set_unordered< T >::find (U && *t*) [inline]

Find a particular element inside of the container.

Template Parameters

U	is any type which can be cast to type T

Parameters

t	is the value of an element to search for

Returns

iterator to location, or to end of the container

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// dcacadaba
auto letters {'d','c','a','c','a','d','a','b','a'};

dsl::shrinking_set_unordered<char> v(std::begin(letters), std::end(
      letters));
dsl::print(v,"\n");

// Return iterator to character c
auto it = v.find('c');

// print all letters from 'c' until the end.
std::for_each(it, std::end(v), [](char c) { std::cout <<c<<",";});

return 0;
}
```

Should produce output

```
{d,c,a,b}
c,a,b,
```

7.74.3.8 template<typename T> T& desalvo_standard_library::shrinking_set_unordered< T >::operator[] (size_t *index*) [inline]

Random access operator

Parameters

index	is the element index

Returns

the value at the given index

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

dsl::shrinking_set_unordered<int> v {3,1,4,1,5,9,2,6,5,3,5,8};
dsl::print(v,"\n");

std::cout << "First elements: " << v[0] << std::endl;
```

```
std::cout << "Sum of first three elements: " << v[0]+v[1]+v[2] << std::endl;

return 0;
}
```

Should produce output

```
{3,1,4,5,9,2,6,8}
First elements: 3
Sum of first three elements: 8
```

7.74.3.9 template<typename T> template<typename U > void desalvo_standard_library::shrinking_set_-unordered< T >::reinitialize (U && *list*) [inline]

reinitialize using a vector of the same type

Parameters

list	is another collection of objects

```
#include <numeric>
#include "desalvo/shrinking_set.h"
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Create vector
std::vector<int> v {1,9,2,8,3,7,4,6,5,5,4,3,2,1};

// Initialize entries using iterators
dsl::shrinking_set_unordered<int> v2(std::begin(v), std::end(v));

dsl::print(v2, "\n");

// Create new vector of values {-10,-9,...,9,10}
std::vector<int> v3(21);
std::iota(std::begin(v3), std::end(v3), -10);

// Reinitialize shrinking set with those values
v2.reinitialize(v3);

dsl::print(v2, "\n");

return 0;
}
```

Should produce output

```
{1,9,2,8,3,7,4,6,5}
{-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10}
```

7.74.3.10 template<typename T> template<typename InputIterator > void desalvo_standard_library-::shrinking_set_unordered< T >::reinitialize (InputIterator *it*, InputIterator *it_stop*) [inline]

reinitialize using another pair of input iterators

Template Parameters

InputIterator	is any input iterator type

Parameters

it	is an iterator referring to the first element
it_stop	is an iterator referring to one after the last element

```
#include "desalvo/shrinking_set.h"
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Create vector
std::vector<int> v {1,9,2,8,3,7,4,6,5,5,4,3,2,1};

// Initialize entries using iterators
dsl::shrinking_set_unordered<int> v2(std::begin(v), std::end(v));

dsl::print(v2, "\n");

// Create new vector of values {-10,-9,...,9,10}
std::vector<int> v3 {-10,10,-9,9,-8,8,-7,7,-6,6,-5,5,-4,4};

// Reinitialize shrinking set with those values
v2.reinitialize(std::begin(v3), std::end(v3));

dsl::print(v2, "\n");

return 0;
}
```

Should produce output

```
{1,9,2,8,3,7,4,6,5}
{-10,10,-9,9,-8,8,-7,7,-6,6,-5,5,-4,4}
```

7.74.3.11 template<typename T> template<typename UnaryPredicate > void **desalvo_standard_library::shrinking_set_unordered**< T >::remove_if (UnaryPredicate *pred*) [inline]

Removes all elements which return true when input into the unary predicate function

Template Parameters

UnaryPredicate	is any function object type

Parameters

pred	is an object with a operator(T)->bool function defined
	`#include "desalvo/std_cout.h"`
	`#include "desalvo/shrinking_set.h"`
	
	`namespace dsl = desalvo_standard_library;`
	
	`int main(int argc, const char * argv[]) {`
	
	`dsl::shrinking_set_unordered<int> v {3,1,4,1,5,9,2,6,5,3,5,8};`
	`dsl::print(v,"\n");`
	
	`// Instead of these lines ...`
	`//v.erase(3);`
	`//v.erase(6);`
	`//v.erase(9);`
	
	`v.remove_if( [](int a) { return a%3 == 0;});`
	
	`dsl::print(v,"\n");`
	
	`return 0;`
	`}`
	
	Should produce output
	`{3,1,4,5,9,2,6,8}`
	`{8,1,4,5,2}`

7.74.3.12 template<typename T> void desalvo_standard_library::shrinking_set_unordered< T >::reset (bool *flag* = false) [inline]

Resets the set to have all elements again

Parameters

flag	specifies whether the list should be resorted or left in unsorted order

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Three little ducks went out one day ...
dsl::shrinking_set_unordered<int> v {3,2,1};
dsl::print(v,"\n");

// over the bridge and far away ...
v.erase(3);

// Mother duck said, "Quack quack quack"
// Two little ducks came back
dsl::print(v,"\n");

// Two little ducks went out one day, over the bridge and far away ...
v.erase(1);

// Mother duck said, "Quack quack quack"
// One little duck came back
dsl::print(v,"\n");

// One little duck went out one day, over the bridge and far away ...
v.erase(2);

// Mother duck said, "Quack quack quack"
// No little ducks came back
dsl::print(v,"\n");

// Sad mother duck went out one day, over the bridge and far away ...
// Mother duck said, "Quack quack quack"
v.reset();

// All the little ducks came back! (In possibly different order!)
dsl::print(v,"\n");

return 0;
}
```

Should produce output

```
{3,2,1}
{1,2}
{2}
{}
{2,1,3}
```

7.74.3.13 template<typename T> size_t desalvo_standard_library::shrinking_set_unordered< T >::size ( ) [inline]

returns the size of the set

Returns

the size of the set

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

dsl::shrinking_set_unordered<int> v {3,2,1,4,5,6,7,8,0};
dsl::print(v,"\n");

std::cout << "size of v: "<<v.size() << std::endl;;

v.remove_if([](int a) { return a%3==0;});

std::cout << "After removing all multiples of 3\n";
std::cout << "size of v: "<<v.size() << std::endl;
```

```
return 0;
}
```

Should produce output

```
{0,1,2,3,4,5,6,7,8}
size of v: 9
After removing all multiples of 3
size of v: 6
```

7.74.3.14 template<typename T> void desalvo_standard_library::shrinking_set_unordered< T >::unerase ( ) [inline]

Unerases the element erased.

```
#include "desalvo/std_cout.h"
#include "desalvo/shrinking_set.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

dsl::shrinking_set_unordered<int> v {3,1,4,1,5,9,2,6,5,3,5,8};
dsl::print(v,"\n");

// Instead of these lines ...
//v.erase(3);
//v.erase(6);
//v.erase(9);

v.remove_if( [](int a) { return a%3 == 0;});

dsl::print(v,"\n");

// Oops!  We wanted to keep the 9.
v.unerase();

dsl::print(v,"\n");

return 0;
}
```

Should produce output

```
{3,1,4,5,9,2,6,8}
{8,1,4,5,2}
{8,1,4,5,2,6}
```

7.74.4 Friends And Related Function Documentation

7.74.4.1 template<typename T> std::ostream& operator<< (std::ostream & *out*, const shrinking_set_unordered< T > & *s*) [friend]

output operator, outputs of the form {#,#,...,#}

Parameters

out	is the stream
s	is the collection of objects

Returns

the stream for chaining

```
#include <initializer_list>
#include "desalvo/shrinking_set.h"
#include "desalvo/std_cout.h"
```

```
namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Create default set of size 0;
dsl::shrinking_set<char> v;

// Initialize set with letters a,b,c,d,e,f
// Cannot just use initializer list, since by default each character is a const char*
dsl::shrinking_set_unordered<char> v2(std::vector<char>({'f','e','c','d',
      'a','b'}));

dsl::print(v,"\n");
dsl::print(v2);

return 0;
}
```

Should produce output

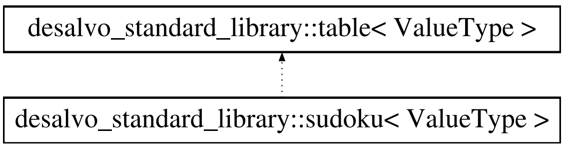
```
{}
{f,e,c,d,a,b}
```

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

- DeSalvo Standard Library/desalvo/shrinking_set.h

7.75 desalvo_standard_library::sudoku< ValueType > Class Template Reference

Inheritance diagram for desalvo_standard_library::sudoku< ValueType >:



Classes

- class block_iterator

Public Member Functions

- **sudoku** (size_t n_input)

Friends

- std::ostream & **operator**<< (std::ostream &out, const sudoku &t)

Additional Inherited Members

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

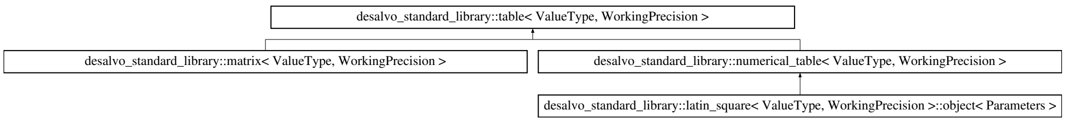
- DeSalvo Standard Library/desalvo/sudoku.h

7.76 desalvo_standard_library::table< ValueType, WorkingPrecision > Class Template Reference

stores a 2-dimensional table of values

```
#include <table.h>
```

Inheritance diagram for desalvo_standard_library::table< ValueType, WorkingPrecision >:



Classes

- class column_const_iterator

    Random Access const_iterator for columns.
- class column_iterator

    Random Access Iterator for columns.
- class const_iterator

    random access const iterator for all entries in table
- class iterator

    random access iterator for a table, treating it like a 1D array
- class row_const_iterator

    Random Access const_iterator for Rows, muentry.
- class row_iterator

    Random Access Iterator for Rows, muentry.
- class table_column_reference

    reference to a column of a table, useful for range-based for loops, C++11
- class table_row_reference

    reference to a row of a table, useful for range-based for loops

Public Member Functions

- table ()
- table (size_t number_of_rows, size_t number_of_columns=1, const ValueType &val=ValueType())
- table (const table &initial_table)
- table (table &&initial_table)
- void swap (table &other)
- table & operator= (table other)
- template< typename VType >
  table (const std::vector< std::vector< VType >> &v)
- template< typename VType >
  table (std::vector< std::vector< VType >> &&v)
- template< typename InputIterator >
  table (size_t number_of_rows, size_t number_of_columns, InputIterator start)
- template< typename RandomAccessIterator >
  table (RandomAccessIterator start, RandomAccessIterator stop)
- template< typename RandomAccessIterator >
  table (RandomAccessIterator start, RandomAccessIterator stop, size_t number_of_columns)
- virtual ∼table ()
- ValueType ∗ get () const

- size_t size_row () const
- size_t size_column () const
- std::pair< size_t, size_t > size () const
- ValueType & operator() (long int i, long int j) const
- bool is_zero () const
- void **set_all_values_to** (const ValueType &new_value)
- ValueType * begin_raw ()
- ValueType * end_raw ()
- ValueType * begin_row_raw (size_t i)
- ValueType * end_row_raw (size_t i)
- const ValueType * cbegin_raw ()
- const ValueType * cend_raw ()
- const ValueType * cbegin_row_raw (size_t i)
- const ValueType * cend_row_raw (size_t i)
- table::const_iterator **cbegin** () const
- table::const_iterator **cend** () const
- table::row_const_iterator cbegin_row (int row) const
- table::row_const_iterator cend_row (int row) const
- table::column_const_iterator cbegin_column (int col) const
- table::column_const_iterator cend_column (int col) const
- table::iterator begin ()
- table::iterator end ()
- table::row_iterator begin_row (int row)
- table::row_iterator end_row (int row)
- table::column_iterator **begin_column** (int col)
- table::column_iterator **end_column** (int col)
- table_row_reference row (int i)
- table_column_reference column (int i)
- template<typename Container = std::vector<ValueType>>
  Container row_sums ()
- template<typename Container = std::vector<ValueType>>
  Container column_sums ()
- void normalize_by_row_sums ()
- void normalize_by_column_sums ()
- void normalize_rows_by_lp (int p=2)
- void normalize_columns_by_lp (int p=2)
- template<typename Container = std::vector<WorkingPrecision>>
  Container row_lp_norms (int p=2)
- template<typename Container = std::vector<WorkingPrecision>>
  Container column_lp_norms (int p=2)
- WorkingPrecision sum ()
- WorkingPrecision mean ()
- WorkingPrecision average ()
- template<typename V , typename Iterator >
  void insert (V &&v, Iterator &&it)
- template<typename V >
  void apply_permutation_map (V &&permutation_map)
- template<typename V >
  void permute_rows (V &&permutation_indices)
- template<typename V >
  void permute_columns (V &&permutation_indices)
- void swap_rows (size_t i, size_t j)
- void swap_columns (size_t i, size_t j)
- template<typename Container = std::vector<ValueType>>
  Container row_as (size_t i)
- template<typename Container = std::vector<ValueType>>
  Container column_as (size_t i)
- std::string as_one_line_matlab_table ()

Friends

- std::ostream & operator<< (std::ostream &out, const table &t)
- bool **operator==** (const table &lhs, const table &rhs)
- bool operator!= (const table::const_iterator &lhs, const table::const_iterator &rhs)
- bool operator<= (const table::const_iterator &lhs, const table::const_iterator &rhs)
- bool operator> (const table::const_iterator &lhs, const table::const_iterator &rhs)
- bool operator>= (const table::const_iterator &lhs, const table::const_iterator &rhs)
- bool operator!= (const table::row_const_iterator &lhs, const table::row_const_iterator &rhs)
- bool operator<= (const table::row_const_iterator &lhs, const table::row_const_iterator &rhs)
- bool operator>= (const table::row_const_iterator &lhs, const table::row_const_iterator &rhs)
- bool operator> (const table::row_const_iterator &lhs, const table::row_const_iterator &rhs)
- bool operator!= (const table::column_const_iterator &lhs, const table::column_const_iterator &rhs)
- bool operator<= (const table::column_const_iterator &lhs, const table::column_const_iterator &rhs)
- bool operator>= (const table::column_const_iterator &lhs, const table::column_const_iterator &rhs)
- bool operator> (const table::column_const_iterator &lhs, const table::column_const_iterator &rhs)
- bool operator!= (const table::iterator &lhs, const table::iterator &rhs)
- bool operator<= (const table::iterator &lhs, const table::iterator &rhs)
- bool operator> (const table::iterator &lhs, const table::iterator &rhs)
- bool operator>= (const table::iterator &lhs, const table::iterator &rhs)
- bool operator!= (const table::row_iterator &lhs, const table::row_iterator &rhs)
- bool operator<= (const table::row_iterator &lhs, const table::row_iterator &rhs)
- bool operator>= (const table::row_iterator &lhs, const table::row_iterator &rhs)
- bool operator> (const table::row_iterator &lhs, const table::row_iterator &rhs)
- bool operator!= (const table::column_iterator &lhs, const table::column_iterator &rhs)
- bool operator<= (const table::column_iterator &lhs, const table::column_iterator &rhs)
- bool operator>= (const table::column_iterator &lhs, const table::column_iterator &rhs)
- bool operator> (const table::column_iterator &lhs, const table::column_iterator &rhs)

7.76.1 Detailed Description

template<typename ValueType = double, typename WorkingPrecision = long double>class desalvo_standard_library::table< ValueType, WorkingPrecision >

stores a 2-dimensional table of values

Stores a contiguous collection of values, organized in a table.

7.76.2 Constructor & Destructor Documentation

7.76.2.1 template<typename ValueType = double, typename WorkingPrecision = long double> desalvo_standard_library::table< ValueType, WorkingPrecision >::table ( ) [inline]

Initializes the "empty" table.

```
#include <iostream>
#include "desalvo/table.h"

namespace dsl = desalvo_standard_library;

// custom class to handle 2D data point, in order to demonstrate how to integrate custom code with existing
       library.
class Point2D {
// output operator: std::cout << pt << std::endl;
friend std::ostream& operator<<(std::ostream& out, const Point2D& pt) {
return out << "(" << pt.x << "," << pt.y << ")";
}

public:
// Initialize value of point
```

```
Point2D(double input_x, double input_y) : x(input_x), y(input_y) { };

// Default constructor, calls more general constructor, new C++11.
//Point2D() : Point2D(0,0) { };
private:
double x, y;
};

int main(int argc, const char * argv[]) {

dsl::table<int> m;
std::cout << m << std::endl;

// Create a 0 x 0 matrix, i.e., empty matrix.  Note that it does not default initialize anything, so the
       underlying type does n't have to have a default constructor defined in this case.  This is similar to making
       an std::set with no elements, it is only until you insert elements that you must have some kind of less than
       function defined.
dsl::table<Point2D> m2;
std::cout << m2 << std::endl;

// Note: Point2D only needs a default constructor when it is default initialized.
// The line below will not compile since the default constructor of Point2D has been commented out.
//dsl::table<Point2D> m3(3,4);

return 0;
}
```

Should produce output

```
{{}}
{{}}
```

7.76.2.2 template< typename ValueType = double, typename WorkingPrecision = long double> desalvo_standard_library::table< ValueType, WorkingPrecision >::table (size_t *number_of_rows*, size_t *number_of_columns* = 1, const ValueType & *val* = ValueType()) [inline]

Initialize entries to value

Parameters

number_of_rows	is the initial number of rows
number_of_columns	is the initial number of columns

val is the initial value for all entries.

```
#include <iostream>
#include "desalvo/table.h"

namespace dsl = desalvo_standard_library;

// custom class to handle 2D data point, in order to demonstrate how to integrate custom code with existin
      library.
class Point2D {
// output operator: std::cout << pt << std::endl;
friend std::ostream& operator<<(std::ostream& out, const Point2D& pt) {
return out << "(" << pt.x << "," << pt.y << ")";
}

public:
// Initialize value of point
Point2D(double input_x, double input_y) : x(input_x), y(input_y) { };

// Default constructor, calls more general constructor, new C++11.
Point2D() : Point2D(0,0) { };
private:
double x, y;
};

int main(int argc, const char * argv[]) {

// Default initializes a 3x1 column of ints
dsl::table<int> m(3);

// Create a 2x2 table of Point2Ds, each default initialized to (0,0)
dsl::table<Point2D> m2(2,2);

// Create a 3x4 table of Point2Ds, each default initialized to (2.718,3.14)
dsl::table<Point2D> m3(3,4, Point2D(2.718,3.14));


std::cout << m << std::endl;
std::cout << m2 << std::endl;
std::cout << m3 << std::endl;


return 0;
}
```

Should produce output

```
{{0},
{0},
{0}}
{{(0,0),(0,0)},
{(0,0),(0,0)}}
{{(2.718,3.14),(2.718,3.14),(2.718,3.14),(2.718,3.14)},
{(2.718,3.14),(2.718,3.14),(2.718,3.14),(2.718,3.14)},
{(2.718,3.14),(2.718,3.14),(2.718,3.14),(2.718,3.14)}}
```

7.76.2.3 template<typename ValueType = double, typename WorkingPrecision = long double> desalvo_standard_library::table< ValueType, WorkingPrecision >::table (const table< ValueType, WorkingPrecision > & *initial_table*) [inline]

copy constructor

Parameters

initial_table is an existing table

```
#include <iostream>
#include "desalvo/table.h"

namespace dsl = desalvo_standard_library;

// custom class to handle 2D data point, in order to demonstrate how to integrate custom code with existin
       library.
class Point2D {
// output operator: std::cout << pt << std::endl;
friend std::ostream& operator<<(std::ostream& out, const Point2D& pt) {
return out << "(" << pt.x << "," << pt.y << ")";
}

public:
// Initialize value of point
Point2D(double input_x, double input_y) : x(input_x), y(input_y) { };

// Default constructor, calls more general constructor, new C++11.
Point2D() : Point2D(0,0) { };
private:
double x, y;
};

int main(int argc, const char * argv[]) {

// Default initializes a 3x1 column of ints
dsl::table<int> m(3);

// Create a 2x2 table of Point2Ds, each default initialized to (0,0)
dsl::table<Point2D> m2(2,2);

// Create a 3x4 table of Point2Ds, each default initialized to (2.718,3.14)
dsl::table<Point2D> m3(3,4, Point2D(2.718,3.14));

// copy construct new tables
auto m4(m);
auto m5(m2);
auto m6(m3);

// Print out values
std::cout << m << std::endl;
std::cout << m2 << std::endl;
std::cout << m3 << std::endl;
std::cout << "Again again!" << std::endl;
std::cout << m4 << std::endl;
std::cout << m5 << std::endl;
std::cout << m6 << std::endl;

return 0;
}
```

Should produce output

```
{{0},
{0},
{0}}
{{(0,0),(0,0)},
{(0,0),(0,0)}}
{{(2.718,3.14),(2.718,3.14),(2.718,3.14),(2.718,3.14)},
{(2.718,3.14),(2.718,3.14),(2.718,3.14),(2.718,3.14)},
{(2.718,3.14),(2.718,3.14),(2.718,3.14),(2.718,3.14)}}
Again again!
{{0},
{0},
{0}}
{{(0,0),(0,0)},
{(0,0),(0,0)}}
{{(2.718,3.14),(2.718,3.14),(2.718,3.14),(2.718,3.14)},
{(2.718,3.14),(2.718,3.14),(2.718,3.14),(2.718,3.14)},
{(2.718,3.14),(2.718,3.14),(2.718,3.14),(2.718,3.14)}}
```

7.76.2.4 template<typename ValueType = double, typename WorkingPrecision = long double> desalvo_standard_library::table< ValueType, WorkingPrecision >::table (table< ValueType, WorkingPrecision > && *initial_table*) [inline]

move constructor

Parameters

initial_table	is an expiring table

```
#include <iostream>
#include "desalvo/table.h"

namespace dsl = desalvo_standard_library;

// custom class to handle 2D data point, in order to demonstrate how to integrate custom code with existir
      library.
class Point2D {
// output operator: std::cout << pt << std::endl;
friend std::ostream& operator<<(std::ostream& out, const Point2D& pt) {
return out << "(" << pt.x << "," << pt.y << ")";
}

public:
// Initialize value of point
Point2D(double input_x, double input_y) : x(input_x), y(input_y) { };

// Default constructor, calls more general constructor, new C++11.
Point2D() : Point2D(0,0) { };
private:
double x, y;
};

// Make a table whose (i,j)-th entry is the value 1./(1.+i+j).  This is called the Hilbert matrix
dsl::table<double> hilbert_table(size_t n) {

// default initialize entries to 1
dsl::table<double> hilbert(n,n,1.);

for(size_t i=0;i<hilbert.size_row();++i)
for(size_t j=0;j<hilbert.size_column();++j)
hilbert(i,j) = 1./(1.+i+j);

return hilbert;
}

int main(int argc, const char * argv[]) {

// Default initializes a 3x1 column of ints
dsl::table<int> m(3);

// Create a 2x2 table of Point2Ds, each default initialized to (0,0)
dsl::table<Point2D> m2(2,2);

// Create a 3x4 table of Point2Ds, each default initialized to (2.718,3.14)
dsl::table<Point2D> m3(3,4, Point2D(2.718,3.14));

// Presumably RVO will be applied here
auto m4(hilbert_table(5));

// move construct new tables
auto m5(std::move(m2));
auto m6(std::move(m3));

// Print out values
std::cout << m << std::endl;

// Since we called move, m2 and m3 are in an undetermined state
//std::cout << m2 << std::endl;
//std::cout << m3 << std::endl;

std::cout << "Hilbert table: " << m4 << std::endl;
std::cout << m5 << std::endl;
std::cout << m6 << std::endl;

return 0;
}
```

Should produce output

```
{{0},
{0},
{0}}
Hilbert table: {{1,0.5,0.333333,0.25,0.2},
{0.5,0.333333,0.25,0.2,0.166667},
{0.333333,0.25,0.2,0.166667,0.142857},
{0.25,0.2,0.166667,0.142857,0.125},
{0.2,0.166667,0.142857,0.125,0.111111}}
{{(0,0),(0,0)},
{(0,0),(0,0)}}
{{(2.718,3.14),(2.718,3.14),(2.718,3.14),(2.718,3.14)},
{(2.718,3.14),(2.718,3.14),(2.718,3.14),(2.718,3.14)},
{(2.718,3.14),(2.718,3.14),(2.718,3.14),(2.718,3.14)}}
```

7.76.2.5 template<typename ValueType = double, typename WorkingPrecision = long double> template<typename VType > desalvo_standard_library::table< ValueType, WorkingPrecision >::table (const std::vector< std::vector< VType >> & v) [inline]

Initialize using a row-major std::array of m*n values

Template Parameters

VType	is any type which can be cast to ValueType

Parameters

init	is an array of values to initilize. Initialize using a vector of vectors

Template Parameters

VType	is any type which can be cast to ValueType

Parameters

v	is an std::vector of std::vector of values to initilize, the first entry of which must be an std::vector with the smallest number of entries as in the other rows.

```
#include <iostream>
#include <valarray>
#include "desalvo/table.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

std::vector<std::vector<int>> sv {{1,2,3},{4,5,6},{7,8,9}};

std::vector<std::string> v1 {"Now", "is", "the"};
std::vector<std::string> v2 {"winter", "of", "our"};
std::vector<std::string> v3 {"discontent", "made", "glorious"};
std::vector<std::string> v4 {"summer", "by", "this"};
std::vector<std::string> v5 {"sonne", "of", "york"};

// Construct a table from an std::vector of std::vectors
dsl::table<int> m(sv);

// Construct a table from a collection of std::vector<std::string>
dsl::table<std::string> richard3( std::vector<std::vector<std::string>>({v1,v2,v3,v4
      ,v5}));

// Print out values
std::cout << m << std::endl;

// Print out some text
std::cout << richard3 << std::endl;

return 0;
}
```

Should produce output

```
{{1,2,3},
{4,5,6},
{7,8,9}}
{{Now,is,the},
{winter,of,our},
{discontent,made,glorious},
{summer,by,this},
{sonne,of,york}}
```

7.76.2.6 template<typename ValueType = double, typename WorkingPrecision = long double> template<typename VType > desalvo_standard_library::table< ValueType, WorkingPrecision >::table (std::vector< std::vector< VType >> && *v*) [inline]

Initialize using a vector of vectors

Template Parameters

VType	is any type which can be cast to ValueType

Parameters

v	is an array of values to initilize.

```
#include <iostream>
#include <vector>
#include "desalvo/table.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

dsl::table<int> m ((std::vector<std::vector<int>>()));
dsl::table<int> m2 ((std::vector<std::vector<int>>(3,{1,2,3})));

// print out empty table
std::cout << m << std::endl;

// print out 3x3 table with each row {1,2,3}
std::cout << m2 << std::endl;

return 0;
}
```

Should produce output

```
{{}}
{{1,2,3},
{1,2,3},
{1,2,3}}
```

7.76.2.7 template<typename ValueType = double, typename WorkingPrecision = long double> template<typename InputIterator > desalvo_standard_library::table< ValueType, WorkingPrecision >::table (size_t *number_of_rows*, size_t *number_of_columns*, InputIterator *start*) [inline]

Initializes a table using dimensions and an iterator at some location with at least as many values as the number of entries in the table.

Template Parameters

InputIterator	is any input iterator

Parameters

number_of_rows	is the number of rows
number_of_-columns	is the number of columns
start	is an input iterator to a starting set of values

```
#include <iostream>
#include <vector>
#include "desalvo/table.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {
```

```
// Make a vector of values
std::vector<int> v {1,2,3,4,5,6,7,8,9,10,11,12};

dsl::table<int> t1(1,12,std::begin(v));
dsl::table<int> t2(2,6,std::begin(v));
dsl::table<int> t3(3,4,std::begin(v));
dsl::table<int> t4(4,3,std::begin(v));
dsl::table<int> t5(6,2,std::begin(v));
dsl::table<int> t6(12,1,std::begin(v));

std::cout << "1x12: \n" << t1 << std::endl << std::endl;
std::cout << "2x6: \n" <<t2 << std::endl << std::endl;
std::cout << "3x4: \n" <<t3 << std::endl << std::endl;
std::cout << "4x3: \n" <<t4 << std::endl << std::endl;
std::cout << "6x2: \n" <<t5 << std::endl << std::endl;
std::cout << "12x1: \n" <<t6 << std::endl << std::endl;

return 0;
}
```

Should produce output

```
1x12:
{{1,2,3,4,5,6,7,8,9,10,11,12}}

2x6:
{{1,2,3,4,5,6},
{7,8,9,10,11,12}}

3x4:
{{1,2,3,4},
{5,6,7,8},
{9,10,11,12}}

4x3:
{{1,2,3},
{4,5,6},
{7,8,9},
{10,11,12}}

6x2:
{{1,2},
{3,4},
{5,6},
{7,8},
{9,10},
{11,12}}

12x1:
{{1},
{2},
{3},
{4},
{5},
{6},
{7},
{8},
{9},
{10},
{11},
{12}}
```

7.76.2.8 template< typename ValueType = double, typename WorkingPrecision = long double > template< typename RandomAccessIterator > desalvo_standard_library::table< ValueType, WorkingPrecision >::table (RandomAccessIterator *start,* RandomAccessIterator *stop*) [inline]

Initializes a 1 x n table using a pair of random access iterators.

Template Parameters

RandomAccessIterator	is any iterator which can take the difference between two of its objects

Parameters

start	is a random access iterator to a starting set of values
stop	is a random access iterator to one after the last value

```
#include "std_cout.h"
#include "table.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

std::vector<size_t> v {1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4};

dsl::table<size_t> t(std::begin(v), std::end(v));

std::cout << t << std::endl;

return 0;
}
```

7.76.2.9 template< typename ValueType = double, typename WorkingPrecision = long double> template< typename RandomAccessIterator > desalvo_standard_library::table< ValueType, WorkingPrecision >::table (RandomAccessIterator *start*, RandomAccessIterator *stop*, size_t *number_of_columns*) [inline]

Initializes a 1 x n table using a pair of random access iterators.

Template Parameters

RandomAccessIterator	is any iterator which can take the difference between two of its objects

Parameters

start	is a random access iterator to a starting set of values
stop	is a random access iterator to one after the last value

```
#include "std_cout.h"
#include "table.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

std::vector<size_t> v {1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4};

dsl::table<size_t> t(std::begin(v), std::end(v));

std::cout << t << std::endl;

return 0;
}
```

7.76.2.10 template< typename ValueType = double, typename WorkingPrecision = long double> virtual desalvo_standard_library::table< ValueType, WorkingPrecision >::~table ( ) [inline], [virtual]

virtual destructors deletes entry memory.

7.76.3 Member Function Documentation

7.76.3.1 template< typename ValueType = double, typename WorkingPrecision = long double> template< typename V > void desalvo_standard_library::table< ValueType, WorkingPrecision >::apply_permutation_map (V && *permutation_map*) [inline]

Applies a permutation to the entries of a table, where the table is treated like a 1D array.

Parameters

permutation_-map	is a permutation of entries.

7.76.3.2 template< typename ValueType = double, typename WorkingPrecision = long double > std::string desalvo_standard_library::table< ValueType, WorkingPrecision >::as_one_line_matlab_table ( ) [inline]

Creates a string so that the table looks like the input for a Matlab array.

Returns

a string of values which can be copied into a Matlab terminal.

7.76.3.3 template< typename ValueType = double, typename WorkingPrecision = long double > WorkingPrecision desalvo_standard_library::table< ValueType, WorkingPrecision >::average ( ) [inline]

Returns the average of all entries. Exists to prevent idiots from not liking the library.

Returns

the average of all entries.

7.76.3.4 template< typename ValueType = double, typename WorkingPrecision = long double > table::iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::begin ( ) [inline]

member begin function so can be used with range-based for loops and other generic algorithms, applied to each element in the table as if it was a 1D array.

Returns

an iterator to the first element in the table.

7.76.3.5 template< typename ValueType = double, typename WorkingPrecision = long double > ValueType* desalvo_standard_library::table< ValueType, WorkingPrecision >::begin_raw ( ) [inline]

returns a pointer to the first element

Returns

a pointer to the first element

```
#include <iostream>
#include <vector>
#include "std_cout.h"
#include "desalvo/table.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Make a vector of values
std::vector<int> v {1,2,3,4,5,6,7,8,9,10,11,12};

// You don't have to use all of the values in v
dsl::table<int> t(3,3,std::begin(v));

std::cout << t << std::endl;

// We now have complete, unfettered access to the elements in t.
auto p = t.begin_raw();
auto p2 = t.end_raw();

// This is useful if we wish to use a routine from another library, e.g., a matrix library, that requires
       two pointers referring to a contiguous array of values.

// This routine rearranges values into the transpose, the last parameter is the size of each row
dsl::transpose(p,p2,3);
```

```
// print out the transposed table
std::cout << t << std::endl;

// const iterators, so can access values but cannot alter them.
auto p3 = t.cbegin_raw();
auto p4 = t.cend_raw();

std::for_each(p3,p4,[](int a) { std::cout << a << ","; }); std::cout<<std::endl;

return 0;
}
```

Should produce output

```
{{1,2,3},
{4,5,6},
{7,8,9}}
{{1,4,7},
{2,5,8},
{3,6,9}}
1,4,7,2,5,8,3,6,9,
```

7.76.3.6 template<typename ValueType = double, typename WorkingPrecision = long double> table::row_iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::begin_row (int *row*) [inline]

Returns an iterator to the first element of a given row

Parameters

row	is the given row

Returns

an iterator to the first element of a given row

7.76.3.7 template<typename ValueType = double, typename WorkingPrecision = long double> ValueType∗ desalvo_standard_library::table< ValueType, WorkingPrecision >::begin_row_raw (size_t *i*) [inline]

Returns a raw pointer to a given row of a table

Parameters

i	is the row number, indexed starting at 0

Returns

a raw pointer to the first element of a given row

```
#include <iostream>
#include <vector>
#include "std_cout.h"
#include "desalvo/table.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Make a vector of values
std::vector<int> v(100);

std::iota(std::begin(v), std::end(v), 0);

// You don't have to use all of the values in v
dsl::table<int> t(10,10,std::begin(v));

std::cout << t << std::endl;
```

```
// Iterate through every other row
for(auto i = 0; i<10; i += 2) {
// Square each value
std::for_each(t.begin_row_raw(i), t.end_row_raw(i), [](int& a) { a *= a; });
}

// Iterate through every third row
for(auto i = 0; i<10; i += 3) {
// Print every other row
std::cout << "{";
std::for_each(t.begin_row_raw(i), t.end_row_raw(i), [](int& a) { std::cout << a<<","; });
std::cout << "}\n";
}

return 0;
}
```

Should produce output

```
{{0,1,2,3,4,5,6,7,8,9},
{10,11,12,13,14,15,16,17,18,19},
{20,21,22,23,24,25,26,27,28,29},
{30,31,32,33,34,35,36,37,38,39},
{40,41,42,43,44,45,46,47,48,49},
{50,51,52,53,54,55,56,57,58,59},
{60,61,62,63,64,65,66,67,68,69},
{70,71,72,73,74,75,76,77,78,79},
{80,81,82,83,84,85,86,87,88,89},
{90,91,92,93,94,95,96,97,98,99}}
{0,1,4,9,16,25,36,49,64,81,}
{30,31,32,33,34,35,36,37,38,39,}
{3600,3721,3844,3969,4096,4225,4356,4489,4624,4761,}
{90,91,92,93,94,95,96,97,98,99,}
```

7.76.3.8 template<typename ValueType = double, typename WorkingPrecision = long double> table::column_const_iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::cbegin_column (int *col*) const [inline]

Returns an iterator referring to the first element of a given column

Parameters

col	is the column number to refer to, can be negative ... why? Any good reason?

Returns

an iterator referring to the first element of a given column

7.76.3.9 template<typename ValueType = double, typename WorkingPrecision = long double> const ValueType* desalvo_standard_library::table< ValueType, WorkingPrecision >::cbegin_raw ( ) [inline]

returns a const pointer to the first element

Returns

a const pointer to the first element

```
#include <iostream>
#include <vector>
#include "std_cout.h"
#include "desalvo/table.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Make a vector of values
std::vector<int> v {1,2,3,4,5,6,7,8,9,10,11,12};

// You don't have to use all of the values in v
```

```
dsl::table<int> t(3,3,std::begin(v));

std::cout << t << std::endl;

// We now have complete, unfettered access to the elements in t.
auto p = t.begin_raw();
auto p2 = t.end_raw();

// This is useful if we wish to use a routine from another library, e.g., a matrix library, that requires
      two pointers referring to a contiguous array of values.

// This routine rearranges values into the transpose, the last parameter is the size of each row
dsl::transpose(p,p2,3);

// print out the transposed table
std::cout << t << std::endl;

// const iterators, so can access values but cannot alter them.
auto p3 = t.cbegin_raw();
auto p4 = t.cend_raw();

std::for_each(p3,p4,[](int a) { std::cout << a << ","; }); std::cout<<std::endl;

return 0;
}
```

Should produce output

```
{{1,2,3},
{4,5,6},
{7,8,9}}
{{1,4,7},
{2,5,8},
{3,6,9}}
1,4,7,2,5,8,3,6,9,
```

7.76.3.10 template<typename ValueType = double, typename WorkingPrecision = long double> table::row_const_iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::cbegin_row (int *row*) const [inline]

const row iterator so can be used in const member functions

Parameters

row	is the row number

Returns

an iterator referring to the first element in the indicated row of the table.

7.76.3.11 template<typename ValueType = double, typename WorkingPrecision = long double> const ValueType∗ desalvo_standard_library::table< ValueType, WorkingPrecision >::cbegin_row_raw (size_t *i*) [inline]

Returns a const raw pointer to a given row of a table

Parameters

i	is the row number, indexed starting at 0

Returns

a const raw pointer to the first element of a given row

```
#include <iostream>
#include <vector>
#include "std_cout.h"
#include "desalvo/table.h"
```

```
namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Make a vector of values
std::vector<int> v(100);

std::iota(std::begin(v), std::end(v), 0);

// You don't have to use all of the values in v
dsl::table<int> t(10,10,std::begin(v));

std::cout << t << std::endl;

// Iterate through every other row
for(auto i = 0; i<10; i += 2) {
// Square each value
std::for_each(t.begin_row_raw(i), t.end_row_raw(i), [](int& a) { a *= a; });
}

// Iterate through every third row
for(auto i = 0; i<10; i += 3) {
// Print every other row
std::cout << "{";
std::for_each(t.cbegin_row_raw(i), t.cend_row_raw(i), [](int a) { std::cout << a<<","; });
std::cout << "}\n";
}

return 0;
}
```

Should produce output

```
{{0,1,2,3,4,5,6,7,8,9},
{10,11,12,13,14,15,16,17,18,19},
{20,21,22,23,24,25,26,27,28,29},
{30,31,32,33,34,35,36,37,38,39},
{40,41,42,43,44,45,46,47,48,49},
{50,51,52,53,54,55,56,57,58,59},
{60,61,62,63,64,65,66,67,68,69},
{70,71,72,73,74,75,76,77,78,79},
{80,81,82,83,84,85,86,87,88,89},
{90,91,92,93,94,95,96,97,98,99}}
{0,1,4,9,16,25,36,49,64,81,}
{30,31,32,33,34,35,36,37,38,39,}
{3600,3721,3844,3969,4096,4225,4356,4489,4624,4761,}
{90,91,92,93,94,95,96,97,98,99,}
```

7.76.3.12 template<typename ValueType = double, typename WorkingPrecision = long double> table::column_const_iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::cend_column (int *col*) const [inline]

Returns an iterator referring to one after the last element of a given column

Parameters

col	is the column number to refer to, can be negative ... why? Any good reason?

Returns

an iterator referring to one after the last element of a given column

7.76.3.13 template<typename ValueType = double, typename WorkingPrecision = long double> const ValueType* desalvo_standard_library::table< ValueType, WorkingPrecision >::cend_raw ( ) [inline]

returns a const pointer to one after the last element

Returns

a const pointer to one after the last element

```
#include <iostream>
#include <vector>
#include "std_cout.h"
#include "desalvo/table.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Make a vector of values
std::vector<int> v {1,2,3,4,5,6,7,8,9,10,11,12};

// You don't have to use all of the values in v
dsl::table<int> t(3,3,std::begin(v));

std::cout << t << std::endl;

// We now have complete, unfettered access to the elements in t.
auto p = t.begin_raw();
auto p2 = t.end_raw();

// This is useful if we wish to use a routine from another library, e.g., a matrix library, that requires
       two pointers referring to a contiguous array of values.

// This routine rearranges values into the transpose, the last parameter is the size of each row
dsl::transpose(p,p2,3);

// print out the transposed table
std::cout << t << std::endl;

// const iterators, so can access values but cannot alter them.
auto p3 = t.cbegin_raw();
auto p4 = t.cend_raw();

std::for_each(p3,p4,[](int a) { std::cout << a << ","; }); std::cout<<std::endl;

return 0;
}
```

Should produce output

```
{{1,2,3},
{4,5,6},
{7,8,9}}
{{1,4,7},
{2,5,8},
{3,6,9}}
1,4,7,2,5,8,3,6,9,
```

7.76.3.14 template<typename ValueType = double, typename WorkingPrecision = long double> table::row_const_iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::cend_row (int *row*) const [inline]

const row iterator so can be used in const member functions

Parameters

row	is the row number

Returns

an iterator referring to one after the last element in the indicated row of the table.

7.76.3.15 template<typename ValueType = double, typename WorkingPrecision = long double> const ValueType* desalvo_standard_library::table< ValueType, WorkingPrecision >::cend_row_raw (size_t *i*) [inline]

Returns a const raw pointer to one after the last element of a given row of a table

Parameters

i	is the row number, indexed starting at 0

Returns

a const raw pointer to one after the last element of a given row

```
#include <iostream>
#include <vector>
#include "std_cout.h"
#include "desalvo/table.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Make a vector of values
std::vector<int> v(100);

std::iota(std::begin(v), std::end(v), 0);

// You don't have to use all of the values in v
dsl::table<int> t(10,10,std::begin(v));

std::cout << t << std::endl;

// Iterate through every other row
for(auto i = 0; i<10; i += 2) {
// Square each value
std::for_each(t.begin_row_raw(i), t.end_row_raw(i), [](int& a) { a *= a; });
}

// Iterate through every third row
for(auto i = 0; i<10; i += 3) {
// Print every other row
std::cout << "{";
std::for_each(t.cbegin_row_raw(i), t.cend_row_raw(i), [](int a) { std::cout << a<<","; });
std::cout << "}\n";
}

return 0;
}
```

Should produce output

```
{{0,1,2,3,4,5,6,7,8,9},
{10,11,12,13,14,15,16,17,18,19},
{20,21,22,23,24,25,26,27,28,29},
{30,31,32,33,34,35,36,37,38,39},
{40,41,42,43,44,45,46,47,48,49},
{50,51,52,53,54,55,56,57,58,59},
{60,61,62,63,64,65,66,67,68,69},
{70,71,72,73,74,75,76,77,78,79},
{80,81,82,83,84,85,86,87,88,89},
{90,91,92,93,94,95,96,97,98,99}}
{0,1,4,9,16,25,36,49,64,81,}
{30,31,32,33,34,35,36,37,38,39,}
{3600,3721,3844,3969,4096,4225,4356,4489,4624,4761,}
{90,91,92,93,94,95,96,97,98,99,}
```

7.76.3.16 template<typename ValueType = double, typename WorkingPrecision = long double> **table_column_reference desalvo_standard_library::table**< ValueType, WorkingPrecision >::column (int *i*) [inline]

Used in range-based for loops.

```
#include <iostream>
#include "desalvo/table.h"
#include "std_cout.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Create 10x10 table, default initialized values (i.e., 0 for int)
dsl::table<int> t(10,10);

for(auto& x : t.column(3))
```

```
    x = 5;

std::cout << t << std::endl;

return 0;
}
```

Should produce output

```
{{0,0,0,5,0,0,0,0,0,0},
{0,0,0,5,0,0,0,0,0,0},
{0,0,0,5,0,0,0,0,0,0},
{0,0,0,5,0,0,0,0,0,0},
{0,0,0,5,0,0,0,0,0,0},
{0,0,0,5,0,0,0,0,0,0},
{0,0,0,5,0,0,0,0,0,0},
{0,0,0,5,0,0,0,0,0,0},
{0,0,0,5,0,0,0,0,0,0},
{0,0,0,5,0,0,0,0,0,0}}
```

Parameters

i	is a column number.

Returns

a reference to a given column of the table.

7.76.3.17 template<typename ValueType = double, typename WorkingPrecision = long double> template<typename Container = std::vector<ValueType>> Container desalvo_standard_library::table< ValueType, WorkingPrecision >::column_as (size_t *i*) [inline]

Create a new container with elements from the entries of a given column, the template parameter specifies the container type, which must supply an input iterator.

Template Parameters

Container	is any class which supplies a .begin() member function and an input iterator.

Parameters

i	is the column

Returns

a container with the given row values stored in the container.

7.76.3.18 template<typename ValueType = double, typename WorkingPrecision = long double> template<typename Container = std::vector<WorkingPrecision>> Container desalvo_standard_library::table< ValueType, WorkingPrecision >::column_lp_norms (int *p* = 2) [inline]

Calculate the l_p norms of each column

Parameters

p	is the norm parameter

Returns

a container with the l_p norms of each column, by default an std::vector

7.76.3.19 template< typename ValueType = double, typename WorkingPrecision = long double > template< typename Container = std::vector< ValueType > > Container desalvo_standard_library::table< ValueType, WorkingPrecision >::column_sums ( ) [inline]

Computes the row sums,

Returns

row sums.

7.76.3.20 template< typename ValueType = double, typename WorkingPrecision = long double > table::iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::end ( ) [inline]

member end function so can be used with range-based for loops and other generic algorithms, applied to each element in the table as if it was a 1D array.

Returns

an iterator to one after the last element in the table.

7.76.3.21 template< typename ValueType = double, typename WorkingPrecision = long double > ValueType∗ desalvo_standard_library::table< ValueType, WorkingPrecision >::end_raw ( ) [inline]

returns a pointer to one after the last element

Returns

a pointer to one after the last element

```
#include <iostream>
#include <vector>
#include "std_cout.h"
#include "desalvo/table.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Make a vector of values
std::vector<int> v {1,2,3,4,5,6,7,8,9,10,11,12};

// You don't have to use all of the values in v
dsl::table<int> t(3,3,std::begin(v));

std::cout << t << std::endl;

// We now have complete, unfettered access to the elements in t.
auto p = t.begin_raw();
auto p2 = t.end_raw();

// This is useful if we wish to use a routine from another library, e.g., a matrix library, that requires
      two pointers referring to a contiguous array of values.

// This routine rearranges values into the transpose, the last parameter is the size of each row
dsl::transpose(p,p2,3);

// print out the transposed table
std::cout << t << std::endl;

// const iterators, so can access values but cannot alter them.
auto p3 = t.cbegin_raw();
auto p4 = t.cend_raw();
```

```
std::for_each(p3,p4,[](int a) { std::cout << a << ","; }); std::cout<<std::endl;

return 0;
}
```

Should produce output

```
{{1,2,3},
{4,5,6},
{7,8,9}}
{{1,4,7},
{2,5,8},
{3,6,9}}
1,4,7,2,5,8,3,6,9,
```

7.76.3.22 template<typename ValueType = double, typename WorkingPrecision = long double> table::row_iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::end_row (int *row*) [inline]

Returns an iterator to one after the last element of a given row

Parameters

row	is the given row

Returns

an iterator to one after the last element of a given row

7.76.3.23 template<typename ValueType = double, typename WorkingPrecision = long double> ValueType* desalvo_standard_library::table< ValueType, WorkingPrecision >::end_row_raw (size_t *i*) [inline]

Returns a raw pointer to one after the last entry of a given row of a table

Parameters

i	is the row number, indexed starting at 0

Returns

a raw pointer to one after the last element of a given row

```
#include <iostream>
#include <vector>
#include "std_cout.h"
#include "desalvo/table.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Make a vector of values
std::vector<int> v(100);

std::iota(std::begin(v), std::end(v), 0);

// You don't have to use all of the values in v
dsl::table<int> t(10,10,std::begin(v));

std::cout << t << std::endl;

// Iterate through every other row
for(auto i = 0; i<10; i += 2) {
// Square each value
std::for_each(t.begin_row_raw(i), t.end_row_raw(i), [](int& a) { a *= a; });
}

// Iterate through every third row
for(auto i = 0; i<10; i += 3) {
```

```
// Print every other row
std::cout << "{";
std::for_each(t.cbegin_row_raw(i), t.cend_row_raw(i), [](int a) { std::cout << a<<","; });
std::cout << "}\n";
}

return 0;
}
```

Should produce output

```
{{0,1,2,3,4,5,6,7,8,9},
{10,11,12,13,14,15,16,17,18,19},
{20,21,22,23,24,25,26,27,28,29},
{30,31,32,33,34,35,36,37,38,39},
{40,41,42,43,44,45,46,47,48,49},
{50,51,52,53,54,55,56,57,58,59},
{60,61,62,63,64,65,66,67,68,69},
{70,71,72,73,74,75,76,77,78,79},
{80,81,82,83,84,85,86,87,88,89},
{90,91,92,93,94,95,96,97,98,99}}
{0,1,4,9,16,25,36,49,64,81,}
{30,31,32,33,34,35,36,37,38,39,}
{3600,3721,3844,3969,4096,4225,4356,4489,4624,4761,}
{90,91,92,93,94,95,96,97,98,99,}
```

7.76.3.24 template< typename ValueType = double, typename WorkingPrecision = long double> ValueType* desalvo_standard_library::table< ValueType, WorkingPrecision >::get ( ) const [inline]

Returns the position of the entry.

Returns

the pointer to the initial value in the entry.

```
#include <iostream>
#include <vector>
#include "desalvo/table.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// make a 3x3 table with rows {1,2,3}
dsl::table<int> m ((std::vector<std::vector<int>>(3,{1,2,3})));

// get a pointer to first element
// VERY DANGEROUS!  This allows you to discretely change the values in m without accessing m directly.
auto p = m.get();

// print out 3x3 table with each row {1,2,3}
std::cout << m << std::endl;

// square each of the values in m
std::for_each(p,p+9,[](int& a) { a*=a;});

// notice there was no mention of m in between these two outputs, and yet ...
std::cout << m << std::endl;

return 0;
}
```

Should produce output

```
{{1,2,3},
{1,2,3},
{1,2,3}}
{{1,4,9},
{1,4,9},
{1,4,9}}
```

7.76.3.25 template< typename ValueType = double, typename WorkingPrecision = long double> template< typename V , typename Iterator > void desalvo_standard_library::table< ValueType, WorkingPrecision >::insert (V && *v*, Iterator && *it*) [inline]

Copies ALL elements of a container starting at a position given by an iterator.

Parameters

v	is the container of elements
it	is an iterator.

7.76.3.26 template<typename ValueType = double, typename WorkingPrecision = long double> bool desalvo_standard_library::table< ValueType, WorkingPrecision >::is_zero ( ) const [inline]

Check if the matrix is all 0s

Returns

true if each entry in the matrix is equivalent to ValueType(0)

```
#include <iostream>
#include <vector>
#include "desalvo/table.h"

namespace dsl = desalvo_standard_library;

// Make a table whose (i,j)-th entry is the value 1./(1.+i+j).  This is called the Hilbert matrix
dsl::table<double> hilbert_table(size_t n) {

// default initialize entries to 1
dsl::table<double> hilbert(n,n,1.);

for(size_t i=0;i<hilbert.size_row();++i)
for(size_t j=0;j<hilbert.size_column();++j)
hilbert(i,j) = 1./(1.+i+j);

return hilbert;
}

int main(int argc, const char * argv[]) {

// make a Hilbert matrix
auto hilbert = hilbert_table(7);

// makes matrix with custom values
dsl::table<int> m(3,4);
m(0,0) = 4;
m(1,2) = 3;
m(2,2) = 7;

// default initialized to all 0s
dsl::table<int> m2(5,5);

std::cout << hilbert.is_zero() << std::endl;
std::cout << m.is_zero() << std::endl;
std::cout << m2.is_zero() << std::endl;

return 0;
}
```

Should produce output

```
0
0
1
```

7.76.3.27 template<typename ValueType = double, typename WorkingPrecision = long double> WorkingPrecision desalvo_standard_library::table< ValueType, WorkingPrecision >::mean ( ) [inline]

Returns the average (i.e., mean) of all entries.

Returns

the average (i.e., mean) of all entries.

7.76.3.28 template<typename ValueType = double, typename WorkingPrecision = long double> void desalvo_standard_library::table< ValueType, WorkingPrecision >::normalize_by_column_sums ( ) [inline]

For each column, calculates the column sums, divides each element by it, so that each column sums to 1.

7.76.3.29 template<typename ValueType = double, typename WorkingPrecision = long double> void desalvo_standard_library::table< ValueType, WorkingPrecision >::normalize_by_row_sums ( ) [inline]

For each row, calculates the row sums, divides each element by it, so that each row sums to 1.

7.76.3.30 template<typename ValueType = double, typename WorkingPrecision = long double> void desalvo_standard_library::table< ValueType, WorkingPrecision >::normalize_columns_by_lp (int *p* = 2) [inline]

Normalize each column by the l_p norm.

Parameters

p	is the norm parameter.

7.76.3.31 template<typename ValueType = double, typename WorkingPrecision = long double> void desalvo_standard_library::table< ValueType, WorkingPrecision >::normalize_rows_by_lp (int *p* = 2) [inline]

Normalize each row by the l_p norm.

Parameters

p	is the norm parameter.

7.76.3.32 template<typename ValueType = double, typename WorkingPrecision = long double> ValueType& desalvo_standard_library::table< ValueType, WorkingPrecision >::operator() (long int *i*, long int *j*) const [inline]

random access of element (i,j) in the table, with indices starting at 0

Parameters

i	is the row number
j	is the column number

Returns

reference to the value in the matrix

```
#include <iostream>
#include <vector>
#include "desalvo/table.h"

namespace dsl = desalvo_standard_library;

// Make a table whose (i,j)-th entry is the value 1./(1.+i+j).  This is called the Hilbert matrix
dsl::table<double> hilbert_table(size_t n) {

// default initialize entries to 1
dsl::table<double> hilbert(n,n,1.);
```

```
for(size_t i=0;i<hilbert.size_row();++i)
for(size_t j=0;j<hilbert.size_column();++j)
hilbert(i,j) = 1./(1.+i+j);

return hilbert;
}

int main(int argc, const char * argv[]) {

// make a Hilbert matrix
auto hilbert = hilbert_table(7);

dsl::table<int> m(3,4);
m(0,0) = 4;
m(1,2) = 3;
m(2,2) = 7;

std::cout << hilbert << std::endl;
std::cout << m << std::endl;


return 0;
}
```

Should produce output

```
{{1,0.5,0.333333,0.25,0.2,0.166667,0.142857},
{0.5,0.333333,0.25,0.2,0.166667,0.142857,0.125},
{0.333333,0.25,0.2,0.166667,0.142857,0.125,0.111111},
{0.25,0.2,0.166667,0.142857,0.125,0.111111,0.1},
{0.2,0.166667,0.142857,0.125,0.111111,0.1,0.0909091},
{0.166667,0.142857,0.125,0.111111,0.1,0.0909091,0.0833333},
{0.142857,0.125,0.111111,0.1,0.0909091,0.0833333,0.0769231}}
{{4,0,0,0},
{0,0,3,0},
{0,0,7,0}}
```

7.76.3.33 template<typename ValueType = double, typename WorkingPrecision = long double> table& desalvo_standard_library::table< ValueType, WorkingPrecision >::operator= (table< ValueType, WorkingPrecision > *other*) [inline]

Assigns values via copy & swap idiom

Parameters

val	is the initial value for all entries.

```
#include <iostream>
#include "desalvo/table.h"

namespace dsl = desalvo_standard_library;

// custom class to handle 2D data point, in order to demonstrate how to integrate custom code with existir
      library.
class Point2D {
// output operator: std::cout << pt << std::endl;
friend std::ostream& operator<<(std::ostream& out, const Point2D& pt) {
return out << "(" << pt.x << "," << pt.y << ")";
}

public:
// Initialize value of point
Point2D(double input_x, double input_y) : x(input_x), y(input_y) { };

// Default constructor, calls more general constructor, new C++11.
Point2D() : Point2D(0,0) { };
private:
double x, y;
};

int main(int argc, const char * argv[]) {

// Default initializes a 3x1 column of ints
dsl::table<int> m(3);

// Create a 2x2 table of Point2Ds, each default initialized to (0,0)
dsl::table<Point2D> m2(2,2);

// Create a 3x4 table of Point2Ds, each default initialized to (2.718,3.14)
dsl::table<Point2D> m3(3,4, Point2D(2.718,3.14));

// Create a 1x3 table with values 3
dsl::table<int> m4(1,3,3);

// Print out values
std::cout << m << std::endl;
std::cout << m2 << std::endl;
std::cout << m3 << std::endl;
std::cout << m4 << std::endl;

// tables are assignable as long as they have same underlying template type, regardless of current size
m = m4;
m2 = m3;

std::cout << "After assignment: \n";
std::cout << m << std::endl;
std::cout << m2 << std::endl;
std::cout << m3 << std::endl;
std::cout << m4 << std::endl;

return 0;
}
```

Should produce output

```
{{0},
{0},
{0}}
{{(0,0),(0,0)},
{(0,0),(0,0)}}
{{(2.718,3.14),(2.718,3.14),(2.718,3.14),(2.718,3.14)},
{(2.718,3.14),(2.718,3.14),(2.718,3.14),(2.718,3.14)},
{(2.718,3.14),(2.718,3.14),(2.718,3.14),(2.718,3.14)}}
{{3,3,3}}
After assignment:
{{3,3,3}}
{{(2.718,3.14),(2.718,3.14),(2.718,3.14),(2.718,3.14)},
{(2.718,3.14),(2.718,3.14),(2.718,3.14),(2.718,3.14)},
{(2.718,3.14),(2.718,3.14),(2.718,3.14),(2.718,3.14)}}
{{(2.718,3.14),(2.718,3.14),(2.718,3.14),(2.718,3.14)},
{(2.718,3.14),(2.718,3.14),(2.718,3.14),(2.718,3.14)},
{(2.718,3.14),(2.718,3.14),(2.718,3.14),(2.718,3.14)}}
{{3,3,3}}
```

7.76.3.34 template<typename ValueType = double, typename WorkingPrecision = long double> template<typename V > void desalvo_standard_library::table< ValueType, WorkingPrecision >::permute_columns (V && *permutation_indices*) [inline]

Permutes columns by a given permutation, assuming the first column is labelled 0.

Parameters

permutation_-indices	is a rearrangement of the numbers 0,1,...,k for which to reorder the columns.

7.76.3.35 template<typename ValueType = double, typename WorkingPrecision = long double> template<typename V > void desalvo_standard_library::table< ValueType, WorkingPrecision >::permute_rows (V && *permutation_indices*) [inline]

Permutes rows by a given permutation, assuming the first row is labelled 0.

Parameters

permutation_-indices	is a rearrangement of the numbers 0,1,...,k for which to reorder the rows.

7.76.3.36 template<typename ValueType = double, typename WorkingPrecision = long double> table_row_reference desalvo_standard_library::table< ValueType, WorkingPrecision >::row (int *i*) [inline]

Used in range-based for loops.

```
#include <iostream>
#include "desalvo/table.h"
#include "std_cout.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Create 10x10 table, default initialized values (i.e., 0 for int)
dsl::table<int> t(10,10);

for(auto& x : t.row(3))
x = 5;

std::cout << t << std::endl;

return 0;
}
```

Should produce output

```
{{0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0},
{5,5,5,5,5,5,5,5,5,5},
{0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0}}
```

Parameters

i	is a row number.

Returns

a reference to a given row of the table.

7.76.3.37 template< typename ValueType = double, typename WorkingPrecision = long double > template< typename Container = std::vector< ValueType > > Container desalvo_standard_library::table< ValueType, WorkingPrecision >::row_as (size_t *i*) [inline]

Create a new container with elements from the entries of a given row, the template parameter specifies the container type, which must supply an input iterator.

Template Parameters

Container	is any class which supplies a .begin() member function and an input iterator.

Parameters

i	is the row

Returns

a container with the given row values stored in the container.

7.76.3.38 template< typename ValueType = double, typename WorkingPrecision = long double > template< typename Container = std::vector< WorkingPrecision > > Container desalvo_standard_library::table< ValueType, WorkingPrecision >::row_lp_norms (int *p* = 2) [inline]

Calculate the l_p norms of each row

Parameters

p	is the norm parameter

Returns

a container with the l_p norms of each row, by default an std::vector

7.76.3.39 template< typename ValueType = double, typename WorkingPrecision = long double > template< typename Container = std::vector< ValueType > > Container desalvo_standard_library::table< ValueType, WorkingPrecision >::row_sums ( ) [inline]

Computes the row sums,

Returns

row sums.

7.76.3.40 template< typename ValueType = double, typename WorkingPrecision = long double > std::pair< size_t,size_t > desalvo_standard_library::table< ValueType, WorkingPrecision >::size ( ) const [inline]

Returns a pair of values corresponding to dimension, i.e., {# rows, #columns}

Returns

a pair {#rows,#columns}

```
#include <iostream>
#include <vector>
#include "desalvo/table.h"

namespace dsl = desalvo_standard_library;

// Make a table whose (i,j)-th entry is the value 1./(1.+i+j).  This is called the Hilbert matrix
dsl::table<double> hilbert_table(size_t n) {

// default initialize entries to 1
dsl::table<double> hilbert(n,n,1.);

for(size_t i=0;i<hilbert.size_row();++i)
for(size_t j=0;j<hilbert.size_column();++j)
hilbert(i,j) = 1./(1.+i+j);

return hilbert;
}

int main(int argc, const char * argv[]) {

// make a Hilbert matrix
auto hilbert = hilbert_table(7);

auto dim = hilbert.size();

std::cout << "We made a " << dim.first << " x " << dim.second << " Hilbert matrix." << std::endl;

std::cout << hilbert << std::endl;


return 0;
}
```

Should produce output

```
We made a 7 x 7 Hilbert matrix.
{{1,0.5,0.333333,0.25,0.2,0.166667,0.142857},
{0.5,0.333333,0.25,0.2,0.166667,0.142857,0.125},
{0.333333,0.25,0.2,0.166667,0.142857,0.125,0.111111},
{0.25,0.2,0.166667,0.142857,0.125,0.111111,0.1},
{0.2,0.166667,0.142857,0.125,0.111111,0.1,0.0909091},
{0.166667,0.142857,0.125,0.111111,0.1,0.0909091,0.0833333},
{0.142857,0.125,0.111111,0.1,0.0909091,0.0833333,0.0769231}}
```

7.76.3.41 template<typename ValueType = double, typename WorkingPrecision = long double> size_t desalvo_standard_library::table< ValueType, WorkingPrecision >::size_column ( ) const [inline]

returns number of columns

Returns

the number of columns

```
#include <iostream>
#include <vector>
#include "desalvo/table.h"

namespace dsl = desalvo_standard_library;

// Make a table whose (i,j)-th entry is the value 1./(1.+i+j).  This is called the Hilbert matrix
dsl::table<double> hilbert_table(size_t n) {

// default initialize entries to 1
dsl::table<double> hilbert(n,n,1.);

for(size_t i=0;i<hilbert.size_row();++i)
for(size_t j=0;j<hilbert.size_column();++j)
hilbert(i,j) = 1./(1.+i+j);

return hilbert;
}

int main(int argc, const char * argv[]) {

// make a Hilbert matrix
```

```
auto hilbert = hilbert_table(7);

// print out the Hilbert matrix as a table of values.
std::cout << hilbert << std::endl;

return 0;
}
```

Should produce output

```
{{1,0.5,0.333333,0.25,0.2,0.166667,0.142857},
{0.5,0.333333,0.25,0.2,0.166667,0.142857,0.125},
{0.333333,0.25,0.2,0.166667,0.142857,0.125,0.111111},
{0.25,0.2,0.166667,0.142857,0.125,0.111111,0.1},
{0.2,0.166667,0.142857,0.125,0.111111,0.1,0.0909091},
{0.166667,0.142857,0.125,0.111111,0.1,0.0909091,0.0833333},
{0.142857,0.125,0.111111,0.1,0.0909091,0.0833333,0.0769231}}
```

7.76.3.42 template<typename ValueType = double, typename WorkingPrecision = long double> size_t desalvo_standard_library::table< ValueType, WorkingPrecision >::size_row ( ) const [inline]

returns number of rows

Returns

the number of rows

```
#include <iostream>
#include <vector>
#include "desalvo/table.h"

namespace dsl = desalvo_standard_library;

// Make a table whose (i,j)-th entry is the value 1./(1.+i+j).  This is called the Hilbert matrix
dsl::table<double> hilbert_table(size_t n) {

// default initialize entries to 1
dsl::table<double> hilbert(n,n,1.);

for(size_t i=0;i<hilbert.size_row();++i)
for(size_t j=0;j<hilbert.size_column();++j)
hilbert(i,j) = 1./(1.+i+j);

return hilbert;
}

int main(int argc, const char * argv[]) {

// make a Hilbert matrix
auto hilbert = hilbert_table(7);

// print out the Hilbert matrix as a table of values.
std::cout << hilbert << std::endl;

return 0;
}
```

Should produce output

```
{{1,0.5,0.333333,0.25,0.2,0.166667,0.142857},
{0.5,0.333333,0.25,0.2,0.166667,0.142857,0.125},
{0.333333,0.25,0.2,0.166667,0.142857,0.125,0.111111},
{0.25,0.2,0.166667,0.142857,0.125,0.111111,0.1},
{0.2,0.166667,0.142857,0.125,0.111111,0.1,0.0909091},
{0.166667,0.142857,0.125,0.111111,0.1,0.0909091,0.0833333},
{0.142857,0.125,0.111111,0.1,0.0909091,0.0833333,0.0769231}}
```

7.76.3.43 template<typename ValueType = double, typename WorkingPrecision = long double> WorkingPrecision desalvo_standard_library::table< ValueType, WorkingPrecision >::sum ( ) [inline]

Returns the sum of all entries.

Returns

the sum of all entries.

7.76.3.44 template<typename ValueType = double, typename WorkingPrecision = long double> void desalvo_standard_library::table< ValueType, WorkingPrecision >::swap (table< ValueType, WorkingPrecision > & *other*) [inline]

Swap two table handles

Parameters

other	table to swap.

```
#include <iostream>
#include "desalvo/table.h"

namespace dsl = desalvo_standard_library;

// custom class to handle 2D data point, in order to demonstrate how to integrate custom code with existir
       library.
class Point2D {
// output operator: std::cout << pt << std::endl;
friend std::ostream& operator<<(std::ostream& out, const Point2D& pt) {
return out << "(" << pt.x << "," << pt.y << ")";
}

public:
// Initialize value of point
Point2D(double input_x, double input_y) : x(input_x), y(input_y) { };

// Default constructor, calls more general constructor, new C++11.
Point2D() : Point2D(0,0) { };
private:
double x, y;
};

int main(int argc, const char * argv[]) {

// Default initializes a 3x1 column of ints
dsl::table<int> m(3);

// Create a 2x2 table of Point2Ds, each default initialized to (0,0)
dsl::table<Point2D> m2(2,2);

// Create a 3x4 table of Point2Ds, each default initialized to (2.718,3.14)
dsl::table<Point2D> m3(3,4, Point2D(2.718,3.14));

// Create a 1x3 table with values 3
dsl::table<int> m4(1,3,3);

// Print out values
std::cout << m << std::endl;
std::cout << m2 << std::endl;
std::cout << m3 << std::endl;
std::cout << m4 << std::endl;

m.swap(m4);
m2.swap(m3);

std::cout << "After swaps: \n";
std::cout << m << std::endl;
std::cout << m2 << std::endl;
std::cout << m3 << std::endl;
std::cout << m4 << std::endl;

return 0;
}
```

Should produce output

```
{{0},
{0},
{0}}
{{(0,0),(0,0)},
{(0,0),(0,0)}}
{{(2.718,3.14),(2.718,3.14),(2.718,3.14),(2.718,3.14)},
{(2.718,3.14),(2.718,3.14),(2.718,3.14),(2.718,3.14)},
{(2.718,3.14),(2.718,3.14),(2.718,3.14),(2.718,3.14)}}
{{3,3,3}}
After swaps:
{{3,3,3}}
{{(2.718,3.14),(2.718,3.14),(2.718,3.14),(2.718,3.14)},
{(2.718,3.14),(2.718,3.14),(2.718,3.14),(2.718,3.14)},
{(2.718,3.14),(2.718,3.14),(2.718,3.14),(2.718,3.14)}}
{{(0,0),(0,0)},
{(0,0),(0,0)}}
{{0},
{0},
{0}}
```

7.76.3.45 template<typename ValueType = double, typename WorkingPrecision = long double> void desalvo_standard_library::table< ValueType, WorkingPrecision >::swap_columns (size_t *i*, size_t *j*) [inline]

Swaps the entries of the two rows indicated by the input parameters.

Parameters

i	is the first row
j	is the second row.

7.76.3.46 template<typename ValueType = double, typename WorkingPrecision = long double> void desalvo_standard_library::table< ValueType, WorkingPrecision >::swap_rows (size_t *i*, size_t *j*) [inline]

Swaps the entries of the two rows indicated by the input parameters.

Parameters

i	is the first row
j	is the second row.

7.76.4 Friends And Related Function Documentation

7.76.4.1 template<typename ValueType = double, typename WorkingPrecision = long double> bool operator!= (const table< ValueType, WorkingPrecision >::const_iterator & *lhs*, const table< ValueType, WorkingPrecision >::const_iterator & *rhs*) [friend]

Tests for iterators in two non-equivalent positions

Parameters

lhs	is the left hand side
rhs	is the right hand side

Returns

true as long as the iterators are not pointing to the exact same coordinates in the exact same table.

7.76.4.2 template<typename ValueType = double, typename WorkingPrecision = long double> bool operator!= (const table< ValueType, WorkingPrecision >::row_const_iterator & *lhs*, const table< ValueType, WorkingPrecision >::row_const_iterator & *rhs*) [friend]

Tests for const_iterators in two equivalent positions

Parameters

t	is the other const_iterator

Returns

true if const_iterators are equivalent

7.76.4.3 template< typename ValueType = double, typename WorkingPrecision = long double> bool operator!= (const table< ValueType, WorkingPrecision >::column_const_iterator & *lhs*, const table< ValueType, WorkingPrecision >::column_const_iterator & *rhs*) [friend]

Tests for const_iterators in two equivalent positions

Parameters

t	is the other const_iterator

Returns

true if const_iterators are equivalent

7.76.4.4 template< typename ValueType = double, typename WorkingPrecision = long double> bool operator!= (const table< ValueType, WorkingPrecision >::iterator & *lhs*, const table< ValueType, WorkingPrecision >::iterator & *rhs*) [friend]

Tests for iterators in two non-equivalent positions

Parameters

lhs	is the left hand side
rhs	is the right hand side

Returns

true as long as the iterators are not pointing to the exact same coordinates in the exact same table.

7.76.4.5 template< typename ValueType = double, typename WorkingPrecision = long double> bool operator!= (const table< ValueType, WorkingPrecision >::row_iterator & *lhs*, const table< ValueType, WorkingPrecision >::row_iterator & *rhs*) [friend]

Tests for iterators in two equivalent positions

Parameters

t	is the other iterator

Returns

true if iterators are equivalent

7.76.4.6 template< typename ValueType = double, typename WorkingPrecision = long double> bool operator!= (const table< ValueType, WorkingPrecision >::column_iterator & *lhs*, const table< ValueType, WorkingPrecision >::column_iterator & *rhs*) [friend]

Tests for iterators in two equivalent positions

Parameters

t	is the other iterator

Returns

true if iterators are equivalent

7.76.4.7 template<typename ValueType = double, typename WorkingPrecision = long double> std::ostream& operator<< (std::ostream & *out*, const table< ValueType, WorkingPrecision > & *t*) [friend]

Output operator, in the form {{#,...,#},{#,...,#},...,{#,..,#}}

Parameters

out	is the stream object
t	is the table object to output

Returns

the stream object for overloading

```
#include <iostream>
#include "desalvo/table.h"

namespace dsl = desalvo_standard_library;

// custom class to handle 2D data point, in order to demonstrate how to integrate custom code with existing
      library.
class Point2D {
// output operator: std::cout << pt << std::endl;
friend std::ostream& operator<<(std::ostream& out, const Point2D& pt) {
return out << "(" << pt.x << "," << pt.y << ")";
}

public:
// Initialize value of point
Point2D(double input_x, double input_y) : x(input_x), y(input_y) { };

// Default constructor, calls more general constructor, new C++11.
Point2D() : Point2D(0,0) { };
private:
double x, y;
};

int main(int argc, const char * argv[]) {

// Create a 0 x 0 matrix, i.e., empty matrix
dsl::table<int> m;
std::cout << m << std::endl;

// Create a 3 x 4 matrix, with entries default initialized
dsl::table<double> m2(3,4);

// Create a 2 x 3 matrix of points.
dsl::table<Point2D> m3(2,3);


std::cout << m2 << std::endl;
std::cout << m3 << std::endl;

return 0;
}
```

Should produce output

```
{{}}
{{0,0,0,0},
{0,0,0,0},
{0,0,0,0}}
{{(0,0),(0,0),(0,0)},
{(0,0),(0,0),(0,0)}}
```

7.76.4.8 template<typename ValueType = double, typename WorkingPrecision = long double> bool operator<= (const table< ValueType, WorkingPrecision >::const_iterator & *lhs*, const table< ValueType, WorkingPrecision >::const_iterator & *rhs*) [friend]

Tests for iterators one weakly less than the other

Parameters

lhs	is the left hand side
rhs	is the right hand side

Returns

true as long as the iterators are pointing to the same table AND the lhs has a smaller row or if equal row then smaller or equal column.

7.76.4.9 template< typename ValueType = double, typename WorkingPrecision = long double > bool operator<= (const table< ValueType, WorkingPrecision >::row_const_iterator & *lhs*, const table< ValueType, WorkingPrecision >::row_const_iterator & *rhs*) [friend]

Tests for const_iterators in same row but strictly smaller column

Parameters

t	is the other const_iterator

Returns

true if const_iterators are equivalent

7.76.4.10 template< typename ValueType = double, typename WorkingPrecision = long double > bool operator<= (const table< ValueType, WorkingPrecision >::column_const_iterator & *lhs*, const table< ValueType, WorkingPrecision >::column_const_iterator & *rhs*) [friend]

Tests for const_iterators in same row but strictly smaller column

Parameters

t	is the other const_iterator

Returns

true if const_iterators are equivalent

7.76.4.11 template< typename ValueType = double, typename WorkingPrecision = long double > bool operator<= (const table< ValueType, WorkingPrecision >::iterator & *lhs*, const table< ValueType, WorkingPrecision >::iterator & *rhs*) [friend]

Tests for iterators one weakly less than the other

Parameters

lhs	is the left hand side
rhs	is the right hand side

Returns

true as long as the iterators are pointing to the same table AND the lhs has a smaller row or if equal row then smaller or equal column.

7.76.4.12 template<typename ValueType = double, typename WorkingPrecision = long double> bool operator<= (const table< ValueType, WorkingPrecision >::row_iterator & *lhs*, const table< ValueType, WorkingPrecision >::row_iterator & *rhs*) [friend]

Tests for iterators in same row but strictly smaller column

Parameters

t	is the other iterator

Returns

true if iterators are equivalent

7.76.4.13 template<typename ValueType = double, typename WorkingPrecision = long double> bool operator<= (const table< ValueType, WorkingPrecision >::column_iterator & *lhs*, const table< ValueType, WorkingPrecision >::column_iterator & *rhs*) [friend]

Tests for iterators in same row but strictly smaller column

Parameters

t	is the other iterator

Returns

true if iterators are equivalent

7.76.4.14 template<typename ValueType = double, typename WorkingPrecision = long double> bool operator> (const table< ValueType, WorkingPrecision >::const_iterator & *lhs*, const table< ValueType, WorkingPrecision >::const_iterator & *rhs*) [friend]

Tests for iterators one strictly greater than the other

Parameters

lhs	is the left hand side
rhs	is the right hand side

Returns

true as long as the const_iterators are pointing to the same table AND the lhs has a smaller row or if equal row then smaller or equal column.

7.76.4.15 template<typename ValueType = double, typename WorkingPrecision = long double> bool operator> (const table< ValueType, WorkingPrecision >::row_const_iterator & *lhs*, const table< ValueType, WorkingPrecision >::row_const_iterator & *rhs*) [friend]

Tests for const_iterators in same row but strictly smaller column

Parameters

t	is the other const_iterator

Returns

true if const_iterators are equivalent

7.76.4.16 template< typename ValueType = double, typename WorkingPrecision = long double> bool operator> (const table< ValueType, WorkingPrecision >::column_const_iterator & *lhs*, const table< ValueType, WorkingPrecision >::column_const_iterator & *rhs*) [friend]

Tests for const_iterators in same row but strictly smaller column

Parameters

t	is the other const_iterator

Returns

true if const_iterators are equivalent

7.76.4.17 template< typename ValueType = double, typename WorkingPrecision = long double> bool operator> (const table< ValueType, WorkingPrecision >::iterator & *lhs*, const table< ValueType, WorkingPrecision >::iterator & *rhs*) [friend]

Tests for iterators one strictly greater than the other

Parameters

lhs	is the left hand side
rhs	is the right hand side

Returns

true as long as the iterators are pointing to the same table AND the lhs has a smaller row or if equal row then smaller or equal column.

7.76.4.18 template< typename ValueType = double, typename WorkingPrecision = long double> bool operator> (const table< ValueType, WorkingPrecision >::row_iterator & *lhs*, const table< ValueType, WorkingPrecision >::row_iterator & *rhs*) [friend]

Tests for iterators in same row but strictly smaller column

Parameters

t	is the other iterator

Returns

true if iterators are equivalent

7.76.4.19 template< typename ValueType = double, typename WorkingPrecision = long double> bool operator> (const table< ValueType, WorkingPrecision >::column_iterator & *lhs*, const table< ValueType, WorkingPrecision >::column_iterator & *rhs*) [friend]

Tests for iterators in same row but strictly smaller column

Parameters

t	is the other iterator

Returns

true if iterators are equivalent

7.76.4.20 template<typename ValueType = double, typename WorkingPrecision = long double> bool operator>= (const table< ValueType, WorkingPrecision >::const_iterator & *lhs*, const table< ValueType, WorkingPrecision >::const_iterator & *rhs*) [friend]

Tests for const_iterators one weakly greater than the other

Parameters

lhs	is the left hand side
rhs	is the right hand side

Returns

true as long as the const_iterators are pointing to the same table AND the lhs has a smaller row or if equal row then smaller or equal column.

7.76.4.21 template<typename ValueType = double, typename WorkingPrecision = long double> bool operator>= (const table< ValueType, WorkingPrecision >::row_const_iterator & *lhs*, const table< ValueType, WorkingPrecision >::row_const_iterator & *rhs*) [friend]

Tests for const_iterators in same row but strictly smaller column

Parameters

t	is the other const_iterator

Returns

true if const_iterators are equivalent

7.76.4.22 template<typename ValueType = double, typename WorkingPrecision = long double> bool operator>= (const table< ValueType, WorkingPrecision >::column_const_iterator & *lhs*, const table< ValueType, WorkingPrecision >::column_const_iterator & *rhs*) [friend]

Tests for const_iterators in same row but strictly smaller column

Parameters

t	is the other const_iterator

Returns

true if const_iterators are equivalent

7.76.4.23 template< typename ValueType = double, typename WorkingPrecision = long double > bool operator>= (const **table**< ValueType, WorkingPrecision >**::iterator** & *lhs*, const **table**< ValueType, WorkingPrecision >**::iterator** & *rhs*) `[friend]`

Tests for iterators one weakly greater than the other

Parameters

lhs	is the left hand side
rhs	is the right hand side

Returns

true as long as the iterators are pointing to the same table AND the lhs has a smaller row or if equal row then smaller or equal column.

7.76.4.24 template< typename ValueType = double, typename WorkingPrecision = long double > bool operator>= (const **table**< ValueType, WorkingPrecision >**::row_iterator** & *lhs*, const **table**< ValueType, WorkingPrecision >**::row_iterator** & *rhs*) `[friend]`

Tests for iterators in same row but strictly smaller column

Parameters

t	is the other iterator

Returns

true if iterators are equivalent

7.76.4.25 template< typename ValueType = double, typename WorkingPrecision = long double > bool operator>= (const **table**< ValueType, WorkingPrecision >**::column_iterator** & *lhs*, const **table**< ValueType, WorkingPrecision >**::column_iterator** & *rhs*) `[friend]`

Tests for iterators in same row but strictly smaller column

Parameters

t	is the other iterator

Returns

true if iterators are equivalent

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

- DeSalvo Standard Library/desalvo/table.h

7.77 desalvo_standard_library::table< ValueType, WorkingPrecision >::table_column_reference Class Reference

reference to a column of a table, useful for range-based for loops, C++11

`#include <table.h>`

Public Member Functions

- table_column_reference (table *m=nullptr, int col=0)
- column_iterator begin () const
- column_iterator end () const

7.77.1 Detailed Description

template<typename ValueType = double, typename WorkingPrecision = long double>class desalvo_standard_library::table< ValueType, WorkingPrecision >::table_column_reference

reference to a column of a table, useful for range-based for loops, C++11

This is to be used to access columns of a table. The object can be used in a range-based for loop.

```
#include <iostream>
#include <vector>
#include "desalvo/table.h"
#include "std_cout.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Create 10x10 table, default initialized values (i.e., 0 for int)
dsl::table<int> t(10,10);

// Very simple linear congruential engine
int A = 16807;
int C = 127;
int value = 1;

// initialize values using custom linear congruential engine
for(auto& i : t)
i = (value = ( (A*value)%C));

std::cout << "t: \n" << t << std::endl << std::endl;

// table_row_reference objects are pretty neat with range-based for loops, C++

// Regenerate entries in the first row.
for(auto& x : t.row(0))
x = (value = ( (A*value)%C));

std::cout << "t (with first row regenerated): \n" << t << std::endl << std::endl;

// Replace 5-th (index 4) column with new values from the linear congruential engine
for(auto& y : t.column(4))
y = (value = ( (A*value)%C));

std::cout << "t (with fifth column regenerated): \n" << t << std::endl << std::endl;

std::cout << "The first two columns are: \n";
// print first two columns side by side
dsl::print_side_by_side(t.column(0), t.column(1));

std::cout << "\nThe last two rows (transposed) are: \n";
// print last two rows side by side in column format
dsl::print_side_by_side(t.row(8), t.row(9));

return 0;
}
```

Should produce output

```
t:
{{43,71,5,88,101,25,59,124,125,41},
{112,117,78,52,77,9,6,4,45,30},
{20,98,23,100,109,115,119,37,67,87},
{58,81,54,36,24,16,53,120,80,11},
{92,19,55,79,95,21,14,94,105,70},
{89,17,96,64,85,99,66,44,114,76},
{93,62,126,84,56,122,39,26,102,68},
{3,2,86,15,10,49,75,50,118,121},
{123,82,97,107,29,104,27,18,12,8},
{90,60,40,69,46,73,91,103,111,74}}
```

```
t (with first row regenerated):
{{7,47,116,35,108,72,48,32,106,113},
{112,117,78,52,77,9,6,4,45,30},
{20,98,23,100,109,115,119,37,67,87},
{58,81,54,36,24,16,53,120,80,11},
{92,19,55,79,95,21,14,94,105,70},
{89,17,96,64,85,99,66,44,114,76},
{93,62,126,84,56,122,39,26,102,68},
{3,2,86,15,10,49,75,50,118,121},
{123,82,97,107,29,104,27,18,12,8},
{90,60,40,69,46,73,91,103,111,74}}

t (with fifth column regenerated):
{{7,47,116,35,33,72,48,32,106,113},
{112,117,78,52,22,9,6,4,45,30},
{20,98,23,100,57,115,119,37,67,87},
{58,81,54,36,38,16,53,120,80,11},
{92,19,55,79,110,21,14,94,105,70},
{89,17,96,64,31,99,66,44,114,76},
{93,62,126,84,63,122,39,26,102,68},
{3,2,86,15,42,49,75,50,118,121},
{123,82,97,107,28,104,27,18,12,8},
{90,60,40,69,61,73,91,103,111,74}}

The first two columns are:
7  47
112  117
20  98
58  81
92  19
89  17
93  62
3  2
123  82
90  60

The last two rows (transposed) are:
123  90
82  60
97  40
107  69
28  61
104  73
27  91
18  103
12  111
8  74
```

7.77.2 Constructor & Destructor Documentation

7.77.2.1 template< typename ValueType = double, typename WorkingPrecision = long double> desalvo_standard_library::table< ValueType, WorkingPrecision >::table_column_reference::table_column_reference (table * *m* = nullptr, int *col* = 0) [inline]

Constructors with default parameters.

With 2 parameters, specifies the table and the column number.

With 1 parameter, specifies the table, column is 0 by default.

With 0 parameters, table is nullptr and column is 0 by default.

Parameters

m	is a table pointer.
r	is the column number for which to reference.

7.77.3 Member Function Documentation

7.77.3.1 template< typename ValueType = double, typename WorkingPrecision = long double> column_iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::table_column_reference::begin ( ) const [inline]

Member begin function so can be used in generic algorithms and range-based for loops.

Returns

iterator referring to the first element of the indicated column of the table.

7.77.3.2 template<typename ValueType = double, typename WorkingPrecision = long double> column_iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::table_column_reference::end ( ) const [inline]

Member end function so can be used in generic algorithms and range-based for loops.

Returns

iterator referring to one after the last element of the indicated column of the table.

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

- DeSalvo Standard Library/desalvo/table.h

7.78 desalvo_standard_library::table< ValueType, WorkingPrecision >::table_row_reference Class Reference

reference to a row of a table, useful for range-based for loops

`#include <table.h>`

Public Member Functions

- table_row_reference (table *m=nullptr, int r=0)
- row_iterator begin () const
- row_iterator end () const
- template<typename V >
  table_row_reference & **operator=** (const V &v)
- template<typename InputIterator >
  table_row_reference & **operator=** (std::pair< InputIterator, InputIterator > iterators)

7.78.1 Detailed Description

template<typename ValueType = double, typename WorkingPrecision = long double>class desalvo_standard_library::table< ValueType, WorkingPrecision >::table_row_reference

reference to a row of a table, useful for range-based for loops

This is to be used to access rows of a table.

```
#include <iostream>
#include <vector>
#include "desalvo/table.h"
#include "std_cout.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

// Create 10x10 table, default initialized values (i.e., 0 for int)
dsl::table<int> t(10,10);

// Very simple linear congruential engine
int A = 16807;
int C = 127;
```

```
int value = 1;

// initialize values using custom linear congruential engine
for(auto& i : t)
i = (value = ( (A*value)%C));

std::cout << "t: \n" << t << std::endl << std::endl;

// table_row_reference objects are pretty neat with range-based for loops, C++

// Regenerate entries in the first row.
for(auto& x : t.row(0))
x = (value = ( (A*value)%C));

std::cout << "t (with first row regenerated): \n" << t << std::endl << std::endl;

// Replace 5-th (index 4) column with new values from the linear congruential engine
for(auto& y : t.column(4))
y = (value = ( (A*value)%C));

std::cout << "t (with fifth column regenerated): \n" << t << std::endl << std::endl;

std::cout << "The first two columns are: \n";
// print first two columns side by side
dsl::print_side_by_side(t.column(0), t.column(1));

std::cout << "\nThe last two rows (transposed) are: \n";
// print last two rows side by side in column format
dsl::print_side_by_side(t.row(8), t.row(9));

return 0;
}
```

Should produce output

```
t:
{{43,71,5,88,101,25,59,124,125,41},
{112,117,78,52,77,9,6,4,45,30},
{20,98,23,100,109,115,119,37,67,87},
{58,81,54,36,24,16,53,120,80,11},
{92,19,55,79,95,21,14,94,105,70},
{89,17,96,64,85,99,66,44,114,76},
{93,62,126,84,56,122,39,26,102,68},
{3,2,86,15,10,49,75,50,118,121},
{123,82,97,107,29,104,27,18,12,8},
{90,60,40,69,46,73,91,103,111,74}}

t (with first row regenerated):
{{7,47,116,35,108,72,48,32,106,113},
{112,117,78,52,77,9,6,4,45,30},
{20,98,23,100,109,115,119,37,67,87},
{58,81,54,36,24,16,53,120,80,11},
{92,19,55,79,95,21,14,94,105,70},
{89,17,96,64,85,99,66,44,114,76},
{93,62,126,84,56,122,39,26,102,68},
{3,2,86,15,10,49,75,50,118,121},
{123,82,97,107,29,104,27,18,12,8},
{90,60,40,69,46,73,91,103,111,74}}

t (with fifth column regenerated):
{{7,47,116,35,33,72,48,32,106,113},
{112,117,78,52,22,9,6,4,45,30},
{20,98,23,100,57,115,119,37,67,87},
{58,81,54,36,38,16,53,120,80,11},
{92,19,55,79,110,21,14,94,105,70},
{89,17,96,64,31,99,66,44,114,76},
{93,62,126,84,63,122,39,26,102,68},
{3,2,86,15,42,49,75,50,118,121},
{123,82,97,107,28,104,27,18,12,8},
{90,60,40,69,61,73,91,103,111,74}}

The first two columns are:
7  47
112  117
20  98
58  81
92  19
89  17
93  62
3  2
123  82
90  60

The last two rows (transposed) are:
123  90
```

```
82  60
97  40
107  69
28  61
104  73
27  91
18  103
12  111
8  74
```

7.78.2 Constructor & Destructor Documentation

7.78.2.1 template<typename ValueType = double, typename WorkingPrecision = long double> desalvo_standard_library::table< ValueType, WorkingPrecision >::table_row_reference::table_row_reference (table * *m* = `nullptr`, int *r* = `0`) `[inline]`

Constructors with default parameters.

With 2 parameters, specifies the table and the row number.

With 1 parameter, specifies the table, row is 0 by default.

With 0 parameters, table is nullptr and row is 0 by default.

Parameters

m	is a table pointer.
r	is the row number for which to reference.

7.78.3 Member Function Documentation

7.78.3.1 template<typename ValueType = double, typename WorkingPrecision = long double> row_iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::table_row_reference::begin ( ) const `[inline]`

Member begin function so can be used in generic algorithms and range-based for loops.

Returns

iterator referring to the first element of the indicated row of the table.

7.78.3.2 template<typename ValueType = double, typename WorkingPrecision = long double> row_iterator desalvo_standard_library::table< ValueType, WorkingPrecision >::table_row_reference::end ( ) const `[inline]`

Member end function so can be used in generic algorithms and range-based for loops.

Returns

iterator referring to one after the last element of the indicated row of the table.

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

- DeSalvo Standard Library/desalvo/table.h

7.79 desalvo_standard_library::time Class Reference

A class for keeping track of timings easily.

`#include <time.h>`

Public Member Functions

- time ()
- void reset ()
- double toc ()

7.79.1 Detailed Description

A class for keeping track of timings easily.

The idea is to do something like dsl::time tic; function(); cout << "function() took "<< tic.toc() << " seconds "<<endl;

The style is admittedly similar to the Matlab style, but this is because they worked out a decent system and I see no reason not to use the similar style, since many people would already be familiar with it.

7.79.2 Constructor & Destructor Documentation

7.79.2.1 desalvo_standard_library::time::time ( )

Default constructor, initializes clock cycles to current number

7.79.3 Member Function Documentation

7.79.3.1 void desalvo_standard_library::time::reset ( )

Resets the clock cycles to current number in program

7.79.3.2 double desalvo_standard_library::time::toc ( )

Calculates total time in seconds that has elapsed

Returns

number of seconds transpired since construction or last call to reset

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

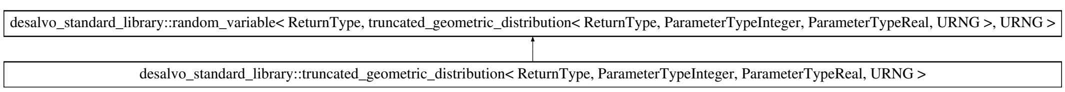
- DeSalvo Standard Library/desalvo/time.h

7.80 desalvo_standard_library::truncated_geometric_distribution< ReturnType, ParameterTypeInteger, ParameterTypeReal, URNG > Class Template Reference

truncated geometric distribution

```
#include <statistics.h>
```

Inheritance diagram for desalvo_standard_library::truncated_geometric_distribution< ReturnType, ParameterTypeInteger, ParameterTypeReal, URNG >:



Public Member Functions

- truncated_geometric_distribution (ParameterTypeInteger input_n, ParameterTypeReal input_q)
- ReturnType **operator()** (URNG &gen)

7.80.1 Detailed Description

template<typename ReturnType = unsigned int, typename ParameterTypeInteger = ReturnType, typename ParameterTypeReal = long double, typename URNG = std::mt19937_64>class desalvo_standard_library::truncated_geometric_distribution< ReturnType, ParameterTypeInteger, ParameterTypeReal, URNG >

truncated geometric distribution

Samples from the distribution P(Z = j) = q^j / (1+q+...+q^(n-1)), where q>0 and n >=2, i.e., a geometric distribution conditioned to be less than n.

7.80.2 Constructor & Destructor Documentation

7.80.2.1 template<typename ReturnType = unsigned int, typename ParameterTypeInteger = ReturnType, typename ParameterTypeReal = long double, typename URNG = std::mt19937_64> desalvo_standard_library::truncated_geometric_distribution< ReturnType, ParameterTypeInteger, ParameterTypeReal, URNG >::truncated_geometric_distribution (ParameterTypeInteger *input_n*, ParameterTypeReal *input_q*) [inline]

Constructs a truncated geometric distribution, where minimum value is 1 and maximum value is n.

P(Z = j) = q^(j-1) / (1+q+...+q^(n-1)), j=1,2,...,n

Parameters

n	is the max possible value
q	is the probability of success

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

- DeSalvo Standard Library/desalvo/statistics.h

Chapter 8

File Documentation

8.1 DeSalvo Standard Library/desalvo/documentation.h File Reference

Contains documentation for files.

Namespaces

- namespace desalvo_standard_library
  think of this namespace like std or boost, I typically use dsl as an alias.

8.1.1 Detailed Description

Contains documentation for files.

Author

Stephen DeSalvo

Date

July, 2015 This is a separate file in order to keep the header files less cluttered, and it gives me the freedom to include as many examples as I wish without worrying about the length of the file.

8.2 DeSalvo Standard Library/desalvo/dsl.h File Reference

includes standard files in desalvo_standard_library

```
#include "file.h"
#include "numerical.h"
#include "std_cout.h"
#include "statistics.h"
#include "shrinking_set.h"
#include "time.h"
```

8.2.1 Detailed Description

includes standard files in desalvo_standard_library This file only includes the standard files in the desalvo_standard_library, but does not define the alias dsl, nor does it define any other keywords. Use dsl_usings.h until you are comfortable with the extended template syntax.

8.3 DeSalvo Standard Library/desalvo/dsl_algorithm.h File Reference

Apply algorithms from the Standard Library to the entire container rather than using iterators.

```
#include <numeric>
#include <functional>
#include <algorithm>
```

Namespaces

- namespace desalvo_standard_library

  think of this namespace like std or boost, I typically use dsl as an alias.

Functions

- template<typename V >
  void desalvo_standard_library::iota (V &v, typename V::value_type val=static_cast< typename V::value_type >(1))
- template<typename V >
  bool desalvo_standard_library::next_permutation (V &v)
- template<typename V , typename Compare >
  bool desalvo_standard_library::next_permutation (V &v, Compare &&cmp)
- template<typename V >
  bool desalvo_standard_library::prev_permutation (V &v)
- template<typename V , typename Compare >
  bool desalvo_standard_library::prev_permutation (V &v, Compare cmp)

8.3.1 Detailed Description

Apply algorithms from the Standard Library to the entire container rather than using iterators. These functions are designed to mimic the Standard Library algorithm file, only instead of using iterators, we apply the transformations to the entire collection of values in the container. Thus, it is intended to have a more Matlab-style, functional feel, with slightly simpler syntax, so that we do not have to keep writing std::begin(v), std::end(v) all the time.

8.4 DeSalvo Standard Library/desalvo/dsl_usings.h File Reference

includes standard files in desalvo_standard_library and includes common aliases and keywords

```
#include "std_cout.h"
#include "file.h"
#include "numerical.h"
#include "dsl_algorithm.h"
#include "statistics.h"
#include "shrinking_set.h"
#include "sequence.h"
#include "permutation.h"
#include "time.h"
#include "combinatorics.h"
#include "table.h"
```

8.4.1 Detailed Description

includes standard files in desalvo_standard_library and includes common aliases and keywords This file includes the standard files in the desalvo_standard_library, defines the alias dsl for short, and it defines many other keywords. Use dsl_usings.h until you are comfortable with the extended template syntax.

TODO: example of

```
// three different ways to create an input file.
desalvo_standard_library::file<desalvo_standard_library::file_type::input>
        file("text.txt");
dsl::file<dsl::file_type::input> file2("text.txt");
dsl::file<input> file3("text.txt");
```

8.5 DeSalvo Standard Library/desalvo/file.h File Reference

Operations on Reading/Writing to files.

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <cstdio>
#include <string>
#include <vector>
```

Classes

- class desalvo_standard_library::file< type >

  Partially specialized for input and output.
- class desalvo_standard_library::file< file_type::input >
- class desalvo_standard_library::file< file_type::output >
- class desalvo_standard_library::file< file_type::console >

Namespaces

- namespace desalvo_standard_library

  think of this namespace like std or boost, I typically use dsl as an alias.

Typedefs

- typedef std::ostream &(∗ desalvo_standard_library::manip1)(std::ostream &)

  abbrev. for type 1 manipulators
- typedef std::basic_ios
  < std::ostream::char_type,
  std::ostream::traits_type > desalvo_standard_library::ios_type

  abbrev. for use with typedef for type 2 manipulators
- typedef ios_type &(∗ desalvo_standard_library::manip2)(ios_type &)

  abbrev. for type 2 manipulators
- typedef std::ios_base &(∗ desalvo_standard_library::manip3)(std::ios_base &)

  abbrev. for type 3 manipulators

Enumerations

- enum desalvo_standard_library::file_type { **input**, **output**, **console** }

Functions

- bool desalvo_standard_library::getline (file< file_type::input > &fin, std::string &s)
- template<typename String >
  bool desalvo_standard_library::getline (file< file_type::console > &fin, String &s)

Variables

- std::string **desalvo_standard_library::path::Teaching** = "/Users/stephendesalvo/Documents/Teaching/"
- std::string **desalvo_standard_library::path::Research** = "/Users/stephendesalvo/Documents/Research/"
- std::string **desalvo_standard_library::path::Permutahedron** = "/Users/stephendesalvo/Documents/Research/Permutahedron Visualization/"

8.5.1 Detailed Description

Operations on Reading/Writing to files.

Author

Stephen DeSalvo

Date

July, 2015 Access to file operations that follows the RAII paradigm. Designed for simple access to file read/write functionality. Uses PTS so easily extendible to other cases like append, append_sequence, etc.

file < input/output/console > f;

8.6 DeSalvo Standard Library/desalvo/latin_square.h File Reference

Classes and functions for Latin squares of all orders.

```
#include "statistics.h"
#include "numerical_table.h"
```

Classes

- class desalvo_standard_library::latin_square< ValueType, WorkingPrecision >
- class desalvo_standard_library::latin_square< ValueType, WorkingPrecision >::object< Parameters >
- class desalvo_standard_library::latin_square< ValueType, WorkingPrecision >::generator< Parameters >
- class desalvo_standard_library::latin_square< ValueType, WorkingPrecision >::object< Parameters >
- class desalvo_standard_library::latin_square< ValueType, WorkingPrecision >::generator< Parameters >
- class desalvo_standard_library::latin_square< ValueType, WorkingPrecision >::iterator_random_access< Parameters >
- class desalvo_standard_library::latin_square< ValueType, WorkingPrecision >::iterator_bidirectional< Parameters >
- class desalvo_standard_library::latin_square< ValueType, WorkingPrecision >::iterator_forward< Parameters >

Namespaces

- namespace desalvo_standard_library

  think of this namespace like std or boost, I typically use dsl as an alias.

Functions

- template<typename ValueType = unsigned int, typename WorkingPrecision = unsigned long int, typename Parameters = std::vector<int>>
  latin_square< ValueType,
  WorkingPrecision >::template
  object< Parameters > **desalvo_standard_library::random_latin_square** (ValueType n)

8.6.1 Detailed Description

Classes and functions for Latin squares of all orders. A Latin square of order n is an $n \times n$ table where each row and each column is a permutation of $1, 2, \ldots, n$.

8.7 DeSalvo Standard Library/desalvo/numerical.h File Reference

Deterministic Algorithms for Numerical Operations.

```
#include <iostream>
#include <vector>
#include <functional>
#include <algorithm>
#include <numeric>
#include <iterator>
#include <set>
#include <cmath>
#include <unordered_map>
```

Classes

- class desalvo_standard_library::NotDivisibleBy

  Creates function objects which check for divisibility.
- class desalvo_standard_library::DivisibleBy

  Creates function objects which check for divisibility.
- class desalvo_standard_library::ArithmeticProgression< T >

  Sequence generator for an arithmetic progression {a, a+r, a+2r, ...}.

Namespaces

- namespace desalvo_standard_library

  think of this namespace like std or boost, I typically use dsl as an alias.
- namespace matlab

  functionality designed to mimic Matlab notation

Typedefs

- typedef unsigned long long **desalvo_standard_library::ull**

Functions

- template<typename Integer , typename UnsignedInteger = Integer>
  UnsignedInteger desalvo_standard_library::gcd (Integer a, Integer b)
- template<typename F = double, typename V = std::vector<F>, typename Size = size_t>
  V desalvo_standard_library::range (Size n, F initial_value=1.)
- template<typename F = double, typename V = std::vector<F>, typename Size = size_t>
  V desalvo_standard_library::constant_array (Size n, F initial_value=1.)
- template<typename Size = size_t, typename V = std::vector<std::pair<Size,Size>>>
  V desalvo_standard_library::table_indices (Size m, Size n, Size initial_value_first=0, Size initial_value_-second=0)
- template<typename V , typename Comparison = std::less<typename V::value_type>>
  void desalvo_standard_library::sort_in_place (V &v, Comparison cmp=std::less< typename V::value_type >())
- template<typename F = double, typename V = std::vector<F>>
  void desalvo_standard_library::partial_sum_in_place (V &v)
- template<typename T = bool, typename Vector = std::vector<T>>
  Vector desalvo_standard_library::binary_row (size_t n, size_t k, T val=true)
- template<typename V >
  void desalvo_standard_library::reverse_in_place (V &v)
- template<typename T , typename F = T>
  F desalvo_standard_library::factorial (T n)
- template<typename T1 , typename T2 , typename F = T1>
  F desalvo_standard_library::nfallingk (T1 n, T2 k)
- template<typename T1 , typename T2 , typename F = T1>
  F desalvo_standard_library::binomial (T1 n, T2 k)
- template<typename T , typename F = T>
  F desalvo_standard_library::choose2 (T n)
- template<typename T , typename F = T>
  F desalvo_standard_library::choose3 (T n)
- template<typename T , typename F = T>
  F desalvo_standard_library::choose4 (T n)
- template<typename N , typename T , typename F = T>
  F desalvo_standard_library::binomial_probability (N n, N k, T p)
- template<typename V , typename C >
  void desalvo_standard_library::print_side_by_side (const V &left, const C &right, const std::string &sep=std-::string(" "), const std::string &endline=std::string("\n"))
- template<typename InputIterator1 , typename InputIterator2 >
  void desalvo_standard_library::print_side_by_side (InputIterator1 start1, InputIterator1 stop, InputIterator2 start2, const std::string &sep=std::string(" "), const std::string &endline=std::string("\n"))
- template<typename ReturnValueType = double, typename IntegerType = long long int, typename DataType = ReturnValueType, type-name InputIterator = typename std::vector<DataType>::iterator>
  ReturnValueType desalvo_standard_library::sum_of_powers (InputIterator start, InputIterator stop, Integer-Type power, DataType initial=0.)
- template<typename T >
  std::vector< std::vector< T > > desalvo_standard_library::permutations (std::vector< T > objects)
- template<typename IntegerType , typename ContainerType = std::vector<IntegerType>>
  ContainerType desalvo_standard_library::int_to_digits (IntegerType a, bool left_to_right=true)
- template<typename IntegerType = int, typename ContainerType = std::vector<IntegerType>>
  IntegerType desalvo_standard_library::digits_to_int (ContainerType digits, bool is_left_to_right=true)
- template<typename Iterator , typename IntegerType >
  bool desalvo_standard_library::is_permutation_of_n (Iterator start, const Iterator &stop, IntegerType n)
- template<typename Container , typename BinaryPredicate = std::equal_to<typename Container::value_type>, typename Comparison = std::less<typename Container::value_type>>
  bool desalvo_standard_library::has_unique_elements (Container elements, BinaryPredicate pred=std::equal-_to< typename Container::value_type >(), Comparison cmp=std::less< typename Container::value_type >())

- template< typename UnsignedIntegers >
  bool desalvo_standard_library::is_unique_uints_max_31 (UnsignedIntegers values)
- template< typename ForwardIterator >
  bool desalvo_standard_library::is_unique_uints_max_31 (ForwardIterator first, ForwardIterator last)
- template< typename V >
  V desalvo_standard_library::conjugate_integer_partition (V v)
- template< typename T , typename RandomAccess >
  void desalvo_standard_library::sort_between (RandomAccess start, RandomAccess stop, T val)
- template< typename T , typename ForwardIterator >
  ForwardIterator desalvo_standard_library::binary_search_iterator (ForwardIterator start, ForwardIterator stop, T &&t)
- template< typename T , typename ForwardIterator >
  ForwardIterator desalvo_standard_library::binary_search_iterator_first (ForwardIterator start, ForwardIterator stop, T &&t)
- template< typename _InputIterator , typename Size , typename _OutputIterator , typename _UnaryOperation >
  void desalvo_standard_library::transform_n (_InputIterator __first, Size __n, _OutputIterator __result, _UnaryOperation __op)
- template< typename _InputIterator1 , typename Size , typename _InputIterator2 , typename _OutputIterator , typename _BinaryOperation >
  void desalvo_standard_library::transform_n (_InputIterator1 __first1, Size __n, _InputIterator2 __first2, _OutputIterator __result, _BinaryOperation __binary_op)
- template< typename InputIterator , typename OutputIterator >
  OutputIterator desalvo_standard_library::unique_copy_nonconsecutive (InputIterator start, InputIterator stop, OutputIterator output)
- template< typename InputIterator , typename OutputIterator , typename BinaryPredicate >
  OutputIterator desalvo_standard_library::unique_copy_nonconsecutive (InputIterator start, InputIterator stop, OutputIterator output, BinaryPredicate bin_op)
- template< class RandomAccessIterator >
  void desalvo_standard_library::transpose (RandomAccessIterator first, RandomAccessIterator last, size_t m)
- template< typename V = std::vector< size_t>>
  V desalvo_standard_library::sieve (size_t n)
- std::vector< std::vector< short > > desalvo_standard_library::multiset_subsets (short n, short k)
- std::vector< std::vector< short > > desalvo_standard_library::unique_multiset_subsets (short n, short k)
- size_t desalvo_standard_library::two_by_two_map (const std::vector< short > &v, const std::vector< std::vector< short >> &possibles)
- size_t **desalvo_standard_library::two_by_two_map** (const std::vector< short > &v, const std::vector< std::pair< std::vector< short >, double >> &possibles)
- std::vector< unsigned int > desalvo_standard_library::fizz_buzz_partition (size_t n)
- template< typename IntType = size_t, typename Container = std::vector< std::vector< IntType>>>
  Container desalvo_standard_library::permutation_as_product_of_cycles (const std::vector< IntType > &permutation)
- template< typename IntType , typename Container = std::vector< std::vector< IntType>>>
  Container desalvo_standard_library::permutation_as_product_of_transpositions (const std::vector< IntType > &permutation)
- template< typename V , typename ReturnType = long double>
  ReturnType **desalvo_standard_library::matlab::sum** (V &&v)
- template< typename V , typename ReturnType = double>
  ReturnType **desalvo_standard_library::matlab::mean** (V &&v)
- template< typename V >
  V **desalvo_standard_library::matlab::sort** (V v)
- template< typename F = double, typename V = std::vector< F>>
  V **desalvo_standard_library::matlab::cumsum** (V &&v)
- template< typename Container >
  Container **desalvo_standard_library::matlab::reverse** (const Container &r)
- std::vector< size_t > desalvo_standard_library::permutation_reduction (std::vector< size_t > vals)

8.7.1 Detailed Description

Deterministic Algorithms for Numerical Operations. Stephen DeSalvo

Date

December, 2014 Contains algorithms and various functions for quick numerical evaluations and common collections of numerical values.

/ TODO: UPDATE CODE EXAMPLES USING dsl

```
// Easier to output std containers
#include "DeSalvoOutputLibrary.h"

#include "DeSalvoNumericalLibrary.h"
namespace dsl = DeSalvoNumericalLibrary;
```

Then in main preface all function/class calls with dsl::

```
// Generate a vector with entries 1,...,10 of type size_t
auto x = dsl::range(10);
```

```
// Overloads of operator<< for standard library containers
#include "std_cout.h"

#include "numerical.h"

int main(int argc, const char * argv[])
{
// Examples of how to use function range(...)

// Generate a vector with entries 1,...,10 of type size_t
auto x = dsl::range(10);
std::cout <<"range(10): "<< x << std::endl;

// Generate a vector with entries 1,...,10 of type int
std::vector<int> vec = dsl::range(10);
std::cout << "range(10): "<< vec << std::endl;

// Generate a vector with entries a,...,a+10-1 of type int
std::vector<int> vec2 = dsl::range(10, 5);
std::cout << "range(10,5): "<<vec2 << std::endl;

// Generate a vector with entries a,...,a+10-1 of type int
std::vector<double> vec3 = dsl::range<double>(10, -4.5);
std::cout << "range<double>(10,-4.5): "<<vec3 << std::endl;

// Generate a list with 10 entries starting at -5.43, incrementing by 1.
std::list<double> lst = dsl::range<double, std::list<double>>(10, -5.43);
std::cout <<"range<double,list<double>>(10,-5.43): "<< lst << std::endl;

// Examples of how to use sort, only works with random access iterators for now
std::vector<int> v {1, -1, 23, -756, 222, 5, 4, -3, 77, 18};
std::cout <<"v = "<< v << std::endl;

// Create new vector with elements sorted
auto v_sorted = dsl::sort(v);
std::cout <<"sort(v): "<< v_sorted << std::endl;

// In place sorting of the container, changes the elements rather than creating new object
auto v_copy = v;
dsl::sort_in_place(v_copy);
std::cout <<"sort_in_place(v_copy): "<< v_copy << std::endl;

std::vector<double> probability_masses {0.1, 0.2, 0.3, 0.2, 0.2};

// Find the cumulative distribution using point probability masses
auto cumulative_distribution = dsl::partial_sum(probability_masses);
std::cout << "partial_sum({.1,.2,.3,.2,.2}): "<<cumulative_distribution << std::endl;

// In place replace with cumulative distribution.
auto probs = probability_masses;
dsl::partial_sum_in_place(probs);
std::cout <<"partial_sum_in_place(...):" << probs << std::endl;


// array of 10 bools, first 3 are true
std::vector<bool> switches = dsl::binary_row(10, 3);
std::cout << "binary_row(10,3): "<<switches << std::endl;
```

```
auto s =dsl::binary_row(10,4,std::string("x"));
std::cout << "binary_row(10,4,string(\"x\")): "<<s <<std::endl;

auto rev = dsl::reverse(switches);
std::cout<<"reverse(binary_row(10,3)): "<<rev << std::endl;

std::cout <<"reverse(range(10.,-0.5)): "<< dsl::reverse(dsl::range(10.,-0.5)) << std::endl;

auto rng = dsl::range(11,-5);
dsl::reverse_in_place(rng);
std::cout <<"reverse_in_place(range(11,-5)): "<< rng << std::endl;

// One list is a vector<int> the other is a list<double>, can mix and match types
auto list1 = dsl::range(11);
auto list2 = dsl::range<double, std::list<double>>(11,-5);
std::cout<<"Print two lists of different types side-by-side one element per row"<<std::endl;
dsl::print_side_by_side(list1, list2,",",";");
std::cout<<std::endl;

std::cout << "binomial(10,7): "<< dsl::binomial(10,7) << std::endl;

// choose2(n) == binomial(n,2)
std::cout << "choose2(10): "<< dsl::choose2(10) << std::endl;

// choose3(n) == binomial(n,3)
std::cout << "choose3(10): "<< dsl::choose3(10) << std::endl;

// choose4(n) == binomial(n,4)
std::cout << "choose4(10): "<< dsl::choose4(10) << std::endl;

auto nums = dsl::range(9);
std::cout << "sum of first 9 integers: "<< dsl::sum_of_powers(std::begin(nums), std::end(
      nums), 0, 1)<<std::endl;
std::cout << "sum_of_powers first 9 squared integers: "<< dsl::sum_of_powers(std::begin(
      nums), std::end(nums), 0, 2)<<std::endl;
std::cout << "sum_of_powers first 9 cubed integers: "<< dsl::sum_of_powers(std::begin(
      nums), std::end(nums), 0, 3)<<std::endl;
std::cout << "sum_of_powers first 9 7th power integers: "<< dsl::sum_of_powers(std::begin
      (nums), std::end(nums), 0, 7)<<std::endl;

std::cout<<"Select any 10 out of 10 things, order matters 10!: "<<
      dsl::factorial(10) << std::endl;
std::cout<<"Select any 4 out of 10 things, order matters (10)_4: "<<
      dsl::nfallingk(10,4) << std::endl;
std::cout<<"Select any 4 out of 10 things, unordered 10 choose 4: "<<
      dsl::binomial(10,4) << std::endl;

std::cout<<"Select any 30 out of 30 things, order matters 30!: "<< dsl::ffactorial(30) << std::endl;
std::cout<<"Select any 17 out of 30 things, order matters (30)_17: "<< dsl::fnfallingk(30,17) << std::endl;
std::cout<<"Select any 24 out of 30 things, unordered 30 choose 24: "<< dsl::fbinomial(30,24) << std::endl;


// All permutations of a collection.
auto perms =dsl::permutations( std::vector<std::string>({"John", "Bob", "Sally"}));
std::cout<<"All rearrangements of {John, Bob, Sally}:\n"<<perms<<std::endl;


auto v = dsl::int_to_digits(10);
cout << v << endl;

std::vector<int> s = {1,2,3,4,5};
cout << dsl::digits_to_int(s) << endl;

// Does not work with strict initializer list.
std::cout << dsl::has_unique_elements( std::vector<int>({1,2,3,4,5,6}) ) << std::endl;
std::cout << dsl::has_unique_elements( std::vector<int>({1,4,3,4,5,6}) ) << std::endl;
std::cout << dsl::has_unique_elements( std::vector<int>({1,3,3,4,5,6}) ) << std::endl;
std::cout << dsl::has_unique_elements( std::vector<int>({1,0,3,-0,5,6}) ) << std::endl;
std::cout << dsl::has_unique_elements( std::vector<int>({1,4,3,-1,5,6}) ) << std::endl;


return 0;
}
```

The output should be:

```
range(10): {1,2,3,4,5,6,7,8,9,10}
range(10): {1,2,3,4,5,6,7,8,9,10}
range(10,5): {5,6,7,8,9,10,11,12,13,14}
range<double>(10,-4.5): {-4.5,-3.5,-2.5,-1.5,-0.5,0.5,1.5,2.5,3.5,4.5}
range<double,list<double>>(10,-5.43): {-5.43,-4.43,-3.43,-2.43,-1.43,-0.43,0.57,1.57,2.57,3.57}
v = {1,-1,23,-756,222,5,4,-3,77,18}
sort(v): {-756,-3,-1,1,4,5,18,23,77,222}
sort_in_place(v_copy): {-756,-3,-1,1,4,5,18,23,77,222}
```

```
partial_sum({.1,.2,.3,.2,.2}): {0.1,0.3,0.6,0.8,1}
partial_sum_in_place(...):{0.1,0.3,0.6,0.8,1}
binary_row(10,3): {1,1,1,0,0,0,0,0,0,0}
binary_row(10,4,string("x")): {x,x,x,x,,,,,,}
reverse(binary_row(10,3)): {0,0,0,0,0,0,0,1,1,1}
reverse(range(10.,-0.5)): {8.5,7.5,6.5,5.5,4.5,3.5,2.5,1.5,0.5,-0.5}
reverse_in_place(range(11,-5)): {5,4,3,2,1,0,-1,-2,-3,-4,-5}
Print two lists of different types side-by-side one element per row
1,-5;2,-4;3,-3;4,-2;5,-1;6,0;7,1;8,2;9,3;10,4;11,5;
binomial(10,7): 120
choose2(10): 45
choose3(10): 120
choose4(10): 210
sum of first 9 integers: 45
sum_of_powers first 9 squared integers: 285
sum_of_powers first 9 cubed integers: 2025
sum_of_powers first 9 7th power integers: 8080425
Select any 10 out of 10 things, order matters 10!: 3628800
Select any 4 out of 10 things, order matters (10)_4: 5040
Select any 4 out of 10 things, unordered 10 choose 4: 210
Select any 30 out of 30 things, order matters 30!: 9.68217e+18
Select any 17 out of 30 things, order matters (30)_17: 3.54097e+18
Select any 24 out of 30 things, unordered 30 choose 24: 593775
All rearrangements of {John, Bob, Sally}:
{{Bob,Sally,John},{Sally,Bob,John},{Sally,John,Bob},{John,Sally,Bob},{Bob,John,Sally},{John,Bob,Sally}}
```

8.8 DeSalvo Standard Library/desalvo/shrinking_set.h File Reference

contains sets which start off with a prescribed set of elements and then shrink

```
#include <iostream>
#include <vector>
#include <functional>
#include <initializer_list>
#include <array>
#include "std_cout.h"
#include "numerical.h"
```

Classes

- class desalvo_standard_library::shrinking_set_unordered< T >

  initialized with a set of objects, then efficiently erases and resets again
- class desalvo_standard_library::shrinking_set< T, Comparison >

  initialized with a set of objects, then efficiently erases and resets again keeping non-erased elements in sorted order

Namespaces

- namespace desalvo_standard_library

  think of this namespace like std or boost, I typically use dsl as an alias.

8.8.1 Detailed Description

contains sets which start off with a prescribed set of elements and then shrink

8.9 DeSalvo Standard Library/desalvo/statistics.h File Reference

Collection of functions and classes for random generation and statistics.

```
#include <random>
#include <chrono>
#include <algorithm>
#include <iterator>
#include <valarray>
#include "numerical.h"
#include "file.h"
```

Classes

- class desalvo_standard_library::random_variable< T, Derived, URNG >
- class desalvo_standard_library::discrete_uniform< ReturnType, ParameterType, URNG >
- class desalvo_standard_library::real_uniform< ReturnType, ParameterType, URNG >
- class desalvo_standard_library::truncated_geometric_distribution< ReturnType, ParameterTypeInteger, ParameterTypeReal, URNG >

    truncated geometric distribution
- class desalvo_standard_library::random_distinct_subset< T, V, URNG >

    Generates a subset of size k from {1,2,...,n}.
- class desalvo_standard_library::numeric_data< T, Container >

    stores collections of numeric data, calculates statistics

Namespaces

- namespace desalvo_standard_library

    think of this namespace like std or boost, I typically use dsl as an alias.

Enumerations

- enum **SimulationMethod** { **BruteForce**, **Boltzmann**, **BoltzmannExact**, **PDCDSH** }

Functions

- template<typename T >
  T desalvo_standard_library::random_integer (T a, T b)
- template<typename T , typename V = std::vector<T>>
  V desalvo_standard_library::random_integer_vector (T a, T b, size_t n)
- template<typename N = std::size_t, typename V = std::vector<N>, typename URNG = std::mt19937_64>
  V desalvo_standard_library::random_permutation (N n, URNG &gen=generator_64)
- template<typename T = bool, typename Vector = std::vector<T>, typename URNG = std::mt19937_64>
  Vector desalvo_standard_library::random_binary_row (size_t n, size_t k, T val=true, URNG &gen=generator_64)
- template<typename N = size_t, typename Float = long double, typename V = std::vector<N>, typename URNG = std::mt19937_64>
  V desalvo_standard_library::random_permutation_mallows_in_mallows_form (N n, Float q, URNG &gen=generator_64)
- template<typename N = size_t, typename Float = long double, typename URNG = std::mt19937_64>
  std::vector< N > desalvo_standard_library::random_permutation_mallows_ordering_construction (N n, Float q, URNG &gen=generator_64)
- template<typename N = size_t, typename Float = long double, typename V = std::vector<N>, typename URNG = std::mt19937_64>
  V desalvo_standard_library::random_permutation_mallows (N n, Float q, URNG &gen=generator_64)
- template<typename N = std::size_t, typename V = std::vector<N>, typename URNG = std::mt19937_64>
  V desalvo_standard_library::random_permutation_shifted (N n, N a, URNG &gen=generator_64)

- template<typename N = std::size_t, typename V = std::vector<N>, typename URNG = std::mt19937_64>
  V desalvo_standard_library::random_permutation_fixed_point_free (N n, URNG &gen=generator_64)
- template<typename URNG = std::mt19937>
  size_t desalvo_standard_library::uniform_size_t (size_t a, size_t b, URNG &gen=generator_32)
- template<typename T = bool, typename Vector = std::vector<T>, typename N = size_t, typename URNG = std::mt19937_64>
  Vector desalvo_standard_library::bernoull_iid_fixedsum (N n, N k, URNG &gen=generator_64)
- template<typename T = bool, typename Vector = std::vector<T>, typename N = size_t, typename URNG = std::mt19937_64>
  Vector desalvo_standard_library::set_n_choose_k (N n, N k, URNG &gen=generator_64)
- template<typename T = bool, typename Vector = std::vector<T>, typename N = size_t, typename URNG = std::mt19937_64>
  Vector desalvo_standard_library::set_2n_choose_n (N n, URNG &gen=generator_64)
- template<typename N = size_t, typename V = std::vector<N>, typename URNG = std::mt19937_64>
  V desalvo_standard_library::partial_permutation_rejection (N n, N k, URNG &gen=generator_64)
- template<typename N = size_t, typename V = std::vector<N>, typename URNG = std::mt19937_64>
  V desalvo_standard_library::partial_permutation (N n, N k, URNG &gen=generator_64)
- template<typename T = bool, typename Vector = std::vector<T>, typename N = size_t, typename Real = double, typename URNG = std::mt19937_64>
  Vector **desalvo_standard_library::set_n_choose_k_repeated** (N n, N k, URNG &gen=generator_64)
- template<typename V = std::vector<bool>, typename T = std::vector<double>, typename N = size_t, typename URNG = std::mt19937_64>
  V desalvo_standard_library::bernoulli_fixedsum_rejection (const T &p, N k, URNG &gen=generator_64)
- template<typename V = std::vector<bool>, typename T = std::valarray<double>, typename N = size_t, typename URNG = std::mt19937_64, typename F = double>
  V desalvo_standard_library::poisson_fixedsum_poisson_process (const T &p, N k, URNG &generator=generator_64)

8.9.1 Detailed Description

Collection of functions and classes for random generation and statistics. Stephen DeSalvo

Date

July, 2015 Example Usage:

```
// Generate a distinct set of three elements from {1,2,...,10}
partial_permutation(10,3);
```

8.10 DeSalvo Standard Library/desalvo/std_cin.h File Reference

Overloads for operator>> for standard library containers.

```
#include <iostream>
#include <valarray>
#include <vector>
#include <list>
#include <algorithm>
#include <set>
#include <string>
#include <initializer_list>
#include <array>
#include <utility>
```

Namespaces

- namespace desalvo_standard_library
  think of this namespace like std or boost, I typically use dsl as an alias.

Functions

- template<typename T , typename F >
  std::istream & operator>> (std::istream &in, std::pair< T, F > &vec)
- template<typename T >
  std::istream & operator>> (std::istream &in, std::vector< T > &vec)
- template<typename T >
  std::istream & **operator**>> (std::istream &in, std::initializer_list< T > &vec)
- template<typename T >
  std::istream & operator>> (std::istream &in, std::multiset< T > &my_list)
- template<typename T >
  std::istream & operator>> (std::istream &in, std::set< T > &my_list)
- template<typename T >
  std::istream & operator>> (std::istream &in, std::valarray< T > &vec)
- template<typename T , size_t n>
  std::istream & operator>> (std::istream &in, std::array< T, n > &vec)
- template<typename T >
  std::istream & operator>> (std::istream &in, std::slice_array< T > &vec)
- template<typename T >
  std::istream & operator>> (std::istream &in, std::list< T > &my_list)
- template<typename T , typename String = std::string>
  void desalvo_standard_library::read (T &container, std::istream &in=std::cin)

Variables

- char **unused**

8.10.1 Detailed Description

Overloads for operator>> for standard library containers.

Author

Stephen DeSalvo

Date

December, 2014 This is a collection of overloads to operator>> that allows for loading in of collections of objects from a file. The default format is {1,2,3,4,5}, that is, elements separated by commas and the entire list enclosed in { }.

Only include this file if you resign yourself to the exact same style as I prefer. Otherwise consider writing your own or waiting for another version to come out which is a bit more general.

8.10.2 Function Documentation

8.10.2.1 template<typename T , typename F > std::istream & operator>> (std::istream & *in*, std::pair< T, F > & *vec*)

Templated function for output of pair<T,F> format.

Template Parameters

T	is the first element type
F	is the second element type

Parameters

out	is the output stream
vec	is the pair<T,F> to output

Returns

the output stream.

```
// Program to generate rectangular version of Hilbert matrix
#include "std_cin.h"
#include "std_cout.h"


namespace dsl = desalvo_standard_library;

// custom class to handle 2D data point, in order to demonstrate how to integrate custom code with existing
       library.
class Point2D {
// output operator: std::cout << pt << std::endl;
friend std::ostream& operator<<(std::ostream& out, const Point2D& pt) {
return out << "(" << pt.x << "," << pt.y << ")";
}

// input operator: std::cin >> pt;
friend std::istream& operator>>(std::istream& in, Point2D& pt) {
char unused;

// format is: (x,y)
// More generally: (char)x(char)y(char)
in >> unused;   // (
in >> pt.x;     //  x
in >> unused;   //   ,
in >> pt.y;     //    y
in >> unused;   //     )

return in;
}
public:
// Initialize value of point
Point2D(double input_x, double input_y) : x(input_x), y(input_y) { };

// Default constructor, calls more general constructor, new C++11.
Point2D() : Point2D(0,0) { };
private:
double x, y;
};

int main(int argc, const char * argv[]) {

std::pair<int, int> my_pair;
std::vector<int> v;
std::set<double> v2;
std::set<int> v3;
std::multiset<int> v4;
std::list<Point2D> v5;
std::valarray<double> v6;
std::array<Point2D,4> v7;

std::cout << "Input values in the same format as the default dsl output: \n";
std::cout << "Insert pair values: ";
std::cin >> my_pair;
std::cout << "Insert vector of ints: ";
std::cin >> v;
std::cout << "Insert set of doubles: ";
std::cin >> v2;
std::cout << "Insert set of ints: ";
std::cin >> v3;
std::cout << "Insert multiset of ints: ";
std::cin >> v4;
std::cout << "Insert list of Point2Ds: ";
std::cin >> v5;
std::cout << "Insert valarray of doubles: ";
std::cin >> v6;
std::cout << "Insert array of 4 Point2Ds: ";
std::cin >> v7;

std::cout << "pair stored as: " << my_pair << std::endl;
std::cout << "vector of ints: " << v << std::endl;
std::cout << "set of doubles: " << v2 << std::endl;
std::cout << "set of ints: " << v3 << std::endl;
std::cout << "multiset of ints: " << v4 << std::endl;
std::cout << "list of Point2Ds: " << v5 << std::endl;
std::cout << "valarray of doubles: " << v6 << std::endl;
std::cout << "array of Point2D: " << v7 << std::endl;
```

```
return 0;
}
```

Should produce output like

```
Input values in the same format as the default dsl output:
Insert pair values: {1,2}
Insert vector of ints: {5,4,2,1}
Insert set of doubles: {1.1,1.1,2.2,2.4,3.7}
Insert set of ints: {1,1,1,1,2,2,2,3,3,4,5}
Insert multiset of ints: {1,1,1,1,2,2,2,3,3,4,5}
Insert list of Point2Ds: {(1,2),(3.4,5.4),(0,0)}
Insert valarray of doubles: {1.1,2.2,3.3,4.4,5.5,6.6}
Insert array of 4 Point2Ds: {(0,0),(-1,1),(-1,-1),(1,-1)}
pair stored as: {1,2}
vector of ints: {5,4,2,1}
set of doubles: {1.1,2.2,2.4,3.7}
set of ints: {1,2,3,4,5}
multiset of ints: {1,1,1,1,2,2,2,3,3,4,5}
list of Point2Ds: {(1,2),(3.4,5.4),(0,0)}
valarray of doubles: {1.1,2.2,3.3,4.4,5.5,6.6}
array of Point2D: {(0,0),(-1,1),(-1,-1),(1,-1)}
```

8.10.2.2 template<typename T > std::istream & operator>> (std::istream & *in*, std::vector< T > & *vec*)

Templated function for output of vector<T> format.

Template Parameters

T	is the type of each element in each of the vectors

Parameters

out	is the output stream
vec	is the vector<T> to output

Returns

the output stream.

```
// Program to generate rectangular version of Hilbert matrix
#include "std_cin.h"
#include "std_cout.h"


namespace dsl = desalvo_standard_library;

// custom class to handle 2D data point, in order to demonstrate how to integrate custom code with existing
       library.
class Point2D {
// output operator: std::cout << pt << std::endl;
friend std::ostream& operator<<(std::ostream& out, const Point2D& pt) {
return out << "(" << pt.x << "," << pt.y << ")";
}

// input operator: std::cin >> pt;
friend std::istream& operator>>(std::istream& in, Point2D& pt) {
char unused;

// format is: (x,y)
// More generally: (char)x(char)y(char)
in >> unused;   // (
in >> pt.x;     //  x
in >> unused;   //   ,
in >> pt.y;     //    y
in >> unused;   //     )

return in;
}
public:
// Initialize value of point
Point2D(double input_x, double input_y) : x(input_x), y(input_y) { };
```

```
// Default constructor, calls more general constructor, new C++11.
Point2D() : Point2D(0,0) { };
private:
double x, y;
};

int main(int argc, const char * argv[]) {

std::pair<int, int> my_pair;
std::vector<int> v;
std::set<double> v2;
std::set<int> v3;
std::multiset<int> v4;
std::list<Point2D> v5;
std::valarray<double> v6;
std::array<Point2D,4> v7;

std::cout << "Input values in the same format as the default dsl output: \n";
std::cout << "Insert pair values: ";
std::cin >> my_pair;
std::cout << "Insert vector of ints: ";
std::cin >> v;
std::cout << "Insert set of doubles: ";
std::cin >> v2;
std::cout << "Insert set of ints: ";
std::cin >> v3;
std::cout << "Insert multiset of ints: ";
std::cin >> v4;
std::cout << "Insert list of Point2Ds: ";
std::cin >> v5;
std::cout << "Insert valarray of doubles: ";
std::cin >> v6;
std::cout << "Insert array of 4 Point2Ds: ";
std::cin >> v7;

std::cout << "pair stored as: " << my_pair << std::endl;
std::cout << "vector of ints: " << v << std::endl;
std::cout << "set of doubles: " << v2 << std::endl;
std::cout << "set of ints: " << v3 << std::endl;
std::cout << "multiset of ints: " << v4 << std::endl;
std::cout << "list of Point2Ds: " << v5 << std::endl;
std::cout << "valarray of doubles: " << v6 << std::endl;
std::cout << "array of Point2D: " << v7 << std::endl;


return 0;
}
```

Should produce output like

```
Input values in the same format as the default dsl output:
Insert pair values: {1,2}
Insert vector of ints: {5,4,2,1}
Insert set of doubles: {1.1,1.1,2.2,2.4,3.7}
Insert set of ints: {1,1,1,1,2,2,2,3,3,4,5}
Insert multiset of ints: {1,1,1,1,2,2,2,3,3,4,5}
Insert list of Point2Ds: {(1,2),(3.4,5.4),(0,0)}
Insert valarray of doubles: {1.1,2.2,3.3,4.4,5.5,6.6}
Insert array of 4 Point2Ds: {(0,0),(-1,1),(-1,-1),(1,-1)}
pair stored as: {1,2}
vector of ints: {5,4,2,1}
set of doubles: {1.1,2.2,2.4,3.7}
set of ints: {1,2,3,4,5}
multiset of ints: {1,1,1,1,2,2,2,3,3,4,5}
list of Point2Ds: {(1,2),(3.4,5.4),(0,0)}
valarray of doubles: {1.1,2.2,3.3,4.4,5.5,6.6}
array of Point2D: {(0,0),(-1,1),(-1,-1),(1,-1)}
```

8.10.2.3 template<typename T > std::istream & operator>> (std::istream & *in*, std::multiset< T > & *my_list*)

Templated function for output of std::multiset<T> format.

Template Parameters

T	is the type of each element

Parameters

out	is the output stream
my_list	is the list<T> to output

Returns

the output stream.

```
// Program to generate rectangular version of Hilbert matrix
#include "std_cin.h"
#include "std_cout.h"


namespace dsl = desalvo_standard_library;

// custom class to handle 2D data point, in order to demonstrate how to integrate custom code with existing
       library.
class Point2D {
// output operator: std::cout << pt << std::endl;
friend std::ostream& operator<<(std::ostream& out, const Point2D& pt) {
return out << "(" << pt.x << "," << pt.y << ")";
}

// input operator: std::cin >> pt;
friend std::istream& operator>>(std::istream& in, Point2D& pt) {
char unused;

// format is: (x,y)
// More generally: (char)x(char)y(char)
in >> unused;   // (
in >> pt.x;     //  x
in >> unused;   //   ,
in >> pt.y;     //    y
in >> unused;   //     )

return in;
}
public:
// Initialize value of point
Point2D(double input_x, double input_y) : x(input_x), y(input_y) { };

// Default constructor, calls more general constructor, new C++11.
Point2D() : Point2D(0,0) { };
private:
double x, y;
};

int main(int argc, const char * argv[]) {

std::pair<int, int> my_pair;
std::vector<int> v;
std::set<double> v2;
std::set<int> v3;
std::multiset<int> v4;
std::list<Point2D> v5;
std::valarray<double> v6;
std::array<Point2D,4> v7;

std::cout << "Input values in the same format as the default dsl output: \n";
std::cout << "Insert pair values: ";
std::cin >> my_pair;
std::cout << "Insert vector of ints: ";
std::cin >> v;
std::cout << "Insert set of doubles: ";
std::cin >> v2;
std::cout << "Insert set of ints: ";
std::cin >> v3;
std::cout << "Insert multiset of ints: ";
std::cin >> v4;
std::cout << "Insert list of Point2Ds: ";
std::cin >> v5;
std::cout << "Insert valarray of doubles: ";
std::cin >> v6;
std::cout << "Insert array of 4 Point2Ds: ";
std::cin >> v7;

std::cout << "pair stored as: " << my_pair << std::endl;
std::cout << "vector of ints: " << v << std::endl;
std::cout << "set of doubles: " << v2 << std::endl;
std::cout << "set of ints: " << v3 << std::endl;
std::cout << "multiset of ints: " << v4 << std::endl;
std::cout << "list of Point2Ds: " << v5 << std::endl;
std::cout << "valarray of doubles: " << v6 << std::endl;
std::cout << "array of Point2D: " << v7 << std::endl;
```

```
return 0;
}
```

Should produce output like

```
Input values in the same format as the default dsl output:
Insert pair values: {1,2}
Insert vector of ints: {5,4,2,1}
Insert set of doubles: {1.1,1.1,2.2,2.4,3.7}
Insert set of ints: {1,1,1,1,2,2,2,3,3,4,5}
Insert multiset of ints: {1,1,1,1,2,2,2,3,3,4,5}
Insert list of Point2Ds: {(1,2),(3.4,5.4),(0,0)}
Insert valarray of doubles: {1.1,2.2,3.3,4.4,5.5,6.6}
Insert array of 4 Point2Ds: {(0,0),(-1,1),(-1,-1),(1,-1)}
pair stored as: {1,2}
vector of ints: {5,4,2,1}
set of doubles: {1.1,2.2,2.4,3.7}
set of ints: {1,2,3,4,5}
multiset of ints: {1,1,1,1,2,2,2,3,3,4,5}
list of Point2Ds: {(1,2),(3.4,5.4),(0,0)}
valarray of doubles: {1.1,2.2,3.3,4.4,5.5,6.6}
array of Point2D: {(0,0),(-1,1),(-1,-1),(1,-1)}
```

8.10.2.4 template<typename T > std::istream & operator>> (std::istream & *in*, std::set< T > & *my_list*)

Templated function for output of std::set<T> format.

Template Parameters

T	is the type of each element

Parameters

out	is the output stream
my_list	is the list<T> to output

Returns

the output stream.

```
// Program to generate rectangular version of Hilbert matrix
#include "std_cin.h"
#include "std_cout.h"


namespace dsl = desalvo_standard_library;

// custom class to handle 2D data point, in order to demonstrate how to integrate custom code with existing
       library.
class Point2D {
// output operator: std::cout << pt << std::endl;
friend std::ostream& operator<<(std::ostream& out, const Point2D& pt) {
return out << "(" << pt.x << "," << pt.y << ")";
}

// input operator: std::cin >> pt;
friend std::istream& operator>>(std::istream& in, Point2D& pt) {
char unused;

// format is: (x,y)
// More generally: (char)x(char)y(char)
in >> unused;   // (
in >> pt.x;     //  x
in >> unused;   //   ,
in >> pt.y;     //    y
in >> unused;   //     )

return in;
}
public:
// Initialize value of point
Point2D(double input_x, double input_y) : x(input_x), y(input_y) { };
```

```
// Default constructor, calls more general constructor, new C++11.
Point2D() : Point2D(0,0) { };
private:
double x, y;
};

int main(int argc, const char * argv[]) {

std::pair<int, int> my_pair;
std::vector<int> v;
std::set<double> v2;
std::set<int> v3;
std::multiset<int> v4;
std::list<Point2D> v5;
std::valarray<double> v6;
std::array<Point2D,4> v7;

std::cout << "Input values in the same format as the default dsl output: \n";
std::cout << "Insert pair values: ";
std::cin >> my_pair;
std::cout << "Insert vector of ints: ";
std::cin >> v;
std::cout << "Insert set of doubles: ";
std::cin >> v2;
std::cout << "Insert set of ints: ";
std::cin >> v3;
std::cout << "Insert multiset of ints: ";
std::cin >> v4;
std::cout << "Insert list of Point2Ds: ";
std::cin >> v5;
std::cout << "Insert valarray of doubles: ";
std::cin >> v6;
std::cout << "Insert array of 4 Point2Ds: ";
std::cin >> v7;

std::cout << "pair stored as: " << my_pair << std::endl;
std::cout << "vector of ints: " << v << std::endl;
std::cout << "set of doubles: " << v2 << std::endl;
std::cout << "set of ints: " << v3 << std::endl;
std::cout << "multiset of ints: " << v4 << std::endl;
std::cout << "list of Point2Ds: " << v5 << std::endl;
std::cout << "valarray of doubles: " << v6 << std::endl;
std::cout << "array of Point2D: " << v7 << std::endl;


return 0;
}
```

Should produce output like

```
Input values in the same format as the default dsl output:
Insert pair values: {1,2}
Insert vector of ints: {5,4,2,1}
Insert set of doubles: {1.1,1.1,2.2,2.4,3.7}
Insert set of ints: {1,1,1,1,2,2,2,3,3,4,5}
Insert multiset of ints: {1,1,1,1,2,2,2,3,3,4,5}
Insert list of Point2Ds: {(1,2),(3.4,5.4),(0,0)}
Insert valarray of doubles: {1.1,2.2,3.3,4.4,5.5,6.6}
Insert array of 4 Point2Ds: {(0,0),(-1,1),(-1,-1),(1,-1)}
pair stored as: {1,2}
vector of ints: {5,4,2,1}
set of doubles: {1.1,2.2,2.4,3.7}
set of ints: {1,2,3,4,5}
multiset of ints: {1,1,1,1,2,2,2,3,3,4,5}
list of Point2Ds: {(1,2),(3.4,5.4),(0,0)}
valarray of doubles: {1.1,2.2,3.3,4.4,5.5,6.6}
array of Point2D: {(0,0),(-1,1),(-1,-1),(1,-1)}
```

8.10.2.5 template<typename T > std::istream & operator>> (std::istream & *in*, std::valarray< T > & *vec*)

Templated function for output of std::valarray<T> format.

Template Parameters

T	is the type of each element in the std::valarray

Parameters

out	is the output stream
vec	is the vector<T> to output

Returns

the output stream.

```
// Program to generate rectangular version of Hilbert matrix
#include "std_cin.h"
#include "std_cout.h"


namespace dsl = desalvo_standard_library;

// custom class to handle 2D data point, in order to demonstrate how to integrate custom code with existing
       library.
class Point2D {
// output operator: std::cout << pt << std::endl;
friend std::ostream& operator<<(std::ostream& out, const Point2D& pt) {
return out << "(" << pt.x << "," << pt.y << ")";
}

// input operator: std::cin >> pt;
friend std::istream& operator>>(std::istream& in, Point2D& pt) {
char unused;

// format is: (x,y)
// More generally: (char)x(char)y(char)
in >> unused;   // (
in >> pt.x;     //  x
in >> unused;   //   ,
in >> pt.y;     //    y
in >> unused;   //     )

return in;
}
public:
// Initialize value of point
Point2D(double input_x, double input_y) : x(input_x), y(input_y) { };

// Default constructor, calls more general constructor, new C++11.
Point2D() : Point2D(0,0) { };
private:
double x, y;
};

int main(int argc, const char * argv[]) {

std::pair<int, int> my_pair;
std::vector<int> v;
std::set<double> v2;
std::set<int> v3;
std::multiset<int> v4;
std::list<Point2D> v5;
std::valarray<double> v6;
std::array<Point2D,4> v7;

std::cout << "Input values in the same format as the default dsl output: \n";
std::cout << "Insert pair values: ";
std::cin >> my_pair;
std::cout << "Insert vector of ints: ";
std::cin >> v;
std::cout << "Insert set of doubles: ";
std::cin >> v2;
std::cout << "Insert set of ints: ";
std::cin >> v3;
std::cout << "Insert multiset of ints: ";
std::cin >> v4;
std::cout << "Insert list of Point2Ds: ";
std::cin >> v5;
std::cout << "Insert valarray of doubles: ";
std::cin >> v6;
std::cout << "Insert array of 4 Point2Ds: ";
std::cin >> v7;

std::cout << "pair stored as: " << my_pair << std::endl;
std::cout << "vector of ints: " << v << std::endl;
std::cout << "set of doubles: " << v2 << std::endl;
std::cout << "set of ints: " << v3 << std::endl;
std::cout << "multiset of ints: " << v4 << std::endl;
std::cout << "list of Point2Ds: " << v5 << std::endl;
std::cout << "valarray of doubles: " << v6 << std::endl;
std::cout << "array of Point2D: " << v7 << std::endl;
```

```
return 0;
}
```

Should produce output like

```
Input values in the same format as the default dsl output:
Insert pair values: {1,2}
Insert vector of ints: {5,4,2,1}
Insert set of doubles: {1.1,1.1,2.2,2.4,3.7}
Insert set of ints: {1,1,1,1,2,2,2,3,3,4,5}
Insert multiset of ints: {1,1,1,1,2,2,2,3,3,4,5}
Insert list of Point2Ds: {(1,2),(3.4,5.4),(0,0)}
Insert valarray of doubles: {1.1,2.2,3.3,4.4,5.5,6.6}
Insert array of 4 Point2Ds: {(0,0),(-1,1),(-1,-1),(1,-1)}
pair stored as: {1,2}
vector of ints: {5,4,2,1}
set of doubles: {1.1,2.2,2.4,3.7}
set of ints: {1,2,3,4,5}
multiset of ints: {1,1,1,1,2,2,2,3,3,4,5}
list of Point2Ds: {(1,2),(3.4,5.4),(0,0)}
valarray of doubles: {1.1,2.2,3.3,4.4,5.5,6.6}
array of Point2D: {(0,0),(-1,1),(-1,-1),(1,-1)}
```

8.10.2.6 template<typename T , size_t n> std::istream & operator>> (std::istream & *in,* std::array< T, n > & *vec*)

Templated function for output of std::array<T> format.

Template Parameters

T	is the type of each element in the std::array
n	is the size of the std::array

Parameters

out	is the output stream
vec	is the array<T> to output

Returns

the output stream.

```
// Program to generate rectangular version of Hilbert matrix
#include "std_cin.h"
#include "std_cout.h"


namespace dsl = desalvo_standard_library;

// custom class to handle 2D data point, in order to demonstrate how to integrate custom code with existing
       library.
class Point2D {
// output operator: std::cout << pt << std::endl;
friend std::ostream& operator<<(std::ostream& out, const Point2D& pt) {
return out << "(" << pt.x << "," << pt.y << ")";
}

// input operator: std::cin >> pt;
friend std::istream& operator>>(std::istream& in, Point2D& pt) {
char unused;

// format is: (x,y)
// More generally: (char)x(char)y(char)
in >> unused;   // (
in >> pt.x;     //  x
in >> unused;   //   ,
in >> pt.y;     //    y
in >> unused;   //     )

return in;
}
public:
// Initialize value of point
Point2D(double input_x, double input_y) : x(input_x), y(input_y) { };
```

```
// Default constructor, calls more general constructor, new C++11.
Point2D() : Point2D(0,0) { };
private:
double x, y;
};

int main(int argc, const char * argv[]) {

std::pair<int, int> my_pair;
std::vector<int> v;
std::set<double> v2;
std::set<int> v3;
std::multiset<int> v4;
std::list<Point2D> v5;
std::valarray<double> v6;
std::array<Point2D,4> v7;

std::cout << "Input values in the same format as the default dsl output: \n";
std::cout << "Insert pair values: ";
std::cin >> my_pair;
std::cout << "Insert vector of ints: ";
std::cin >> v;
std::cout << "Insert set of doubles: ";
std::cin >> v2;
std::cout << "Insert set of ints: ";
std::cin >> v3;
std::cout << "Insert multiset of ints: ";
std::cin >> v4;
std::cout << "Insert list of Point2Ds: ";
std::cin >> v5;
std::cout << "Insert valarray of doubles: ";
std::cin >> v6;
std::cout << "Insert array of 4 Point2Ds: ";
std::cin >> v7;

std::cout << "pair stored as: " << my_pair << std::endl;
std::cout << "vector of ints: " << v << std::endl;
std::cout << "set of doubles: " << v2 << std::endl;
std::cout << "set of ints: " << v3 << std::endl;
std::cout << "multiset of ints: " << v4 << std::endl;
std::cout << "list of Point2Ds: " << v5 << std::endl;
std::cout << "valarray of doubles: " << v6 << std::endl;
std::cout << "array of Point2D: " << v7 << std::endl;


return 0;
}
```

Should produce output like

```
Input values in the same format as the default dsl output:
Insert pair values: {1,2}
Insert vector of ints: {5,4,2,1}
Insert set of doubles: {1.1,1.1,2.2,2.4,3.7}
Insert set of ints: {1,1,1,1,2,2,2,3,3,4,5}
Insert multiset of ints: {1,1,1,1,2,2,2,3,3,4,5}
Insert list of Point2Ds: {(1,2),(3.4,5.4),(0,0)}
Insert valarray of doubles: {1.1,2.2,3.3,4.4,5.5,6.6}
Insert array of 4 Point2Ds: {(0,0),(-1,1),(-1,-1),(1,-1)}
pair stored as: {1,2}
vector of ints: {5,4,2,1}
set of doubles: {1.1,2.2,2.4,3.7}
set of ints: {1,2,3,4,5}
multiset of ints: {1,1,1,1,2,2,2,3,3,4,5}
list of Point2Ds: {(1,2),(3.4,5.4),(0,0)}
valarray of doubles: {1.1,2.2,3.3,4.4,5.5,6.6}
array of Point2D: {(0,0),(-1,1),(-1,-1),(1,-1)}
```

8.10.2.7 template<typename T > std::istream & operator>> (std::istream & *in*, std::slice_array< T > & *vec*)

Templated function for output of std::slice_array<T> format.

Template Parameters

T	is the type of each element

Parameters

out	is the output stream
vec	is the vector<T> to output

Returns

the output stream.

8.10.2.8 template< typename T > std::istream & operator>> (std::istream & *in*, std::list< T > & *my_list*)

Templated function for output of std::list<T> format.

Template Parameters

is	the type of each element

Parameters

out	is the output stream
my_list	is the list<T> to output

Returns

the output stream.

```
// Program to generate rectangular version of Hilbert matrix
#include "std_cin.h"
#include "std_cout.h"


namespace dsl = desalvo_standard_library;

// custom class to handle 2D data point, in order to demonstrate how to integrate custom code with existing
       library.
class Point2D {
// output operator: std::cout << pt << std::endl;
friend std::ostream& operator<<(std::ostream& out, const Point2D& pt) {
return out << "(" << pt.x << "," << pt.y << ")";
}

// input operator: std::cin >> pt;
friend std::istream& operator>>(std::istream& in, Point2D& pt) {
char unused;

// format is: (x,y)
// More generally: (char)x(char)y(char)
in >> unused;   // (
in >> pt.x;     //  x
in >> unused;   //   ,
in >> pt.y;     //    y
in >> unused;   //     )

return in;
}
public:
// Initialize value of point
Point2D(double input_x, double input_y) : x(input_x), y(input_y) { };

// Default constructor, calls more general constructor, new C++11.
Point2D() : Point2D(0,0) { };
private:
double x, y;
};

int main(int argc, const char * argv[]) {

std::pair<int, int> my_pair;
std::vector<int> v;
std::set<double> v2;
std::set<int> v3;
std::multiset<int> v4;
std::list<Point2D> v5;
```

```
std::valarray<double> v6;
std::array<Point2D,4> v7;

std::cout << "Input values in the same format as the default dsl output: \n";
std::cout << "Insert pair values: ";
std::cin >> my_pair;
std::cout << "Insert vector of ints: ";
std::cin >> v;
std::cout << "Insert set of doubles: ";
std::cin >> v2;
std::cout << "Insert set of ints: ";
std::cin >> v3;
std::cout << "Insert multiset of ints: ";
std::cin >> v4;
std::cout << "Insert list of Point2Ds: ";
std::cin >> v5;
std::cout << "Insert valarray of doubles: ";
std::cin >> v6;
std::cout << "Insert array of 4 Point2Ds: ";
std::cin >> v7;

std::cout << "pair stored as: " << my_pair << std::endl;
std::cout << "vector of ints: " << v << std::endl;
std::cout << "set of doubles: " << v2 << std::endl;
std::cout << "set of ints: " << v3 << std::endl;
std::cout << "multiset of ints: " << v4 << std::endl;
std::cout << "list of Point2Ds: " << v5 << std::endl;
std::cout << "valarray of doubles: " << v6 << std::endl;
std::cout << "array of Point2D: " << v7 << std::endl;


return 0;
}
```

Should produce output like

```
Input values in the same format as the default dsl output:
Insert pair values: {1,2}
Insert vector of ints: {5,4,2,1}
Insert set of doubles: {1.1,1.1,2.2,2.4,3.7}
Insert set of ints: {1,1,1,1,2,2,2,3,3,4,5}
Insert multiset of ints: {1,1,1,1,2,2,2,3,3,4,5}
Insert list of Point2Ds: {(1,2),(3.4,5.4),(0,0)}
Insert valarray of doubles: {1.1,2.2,3.3,4.4,5.5,6.6}
Insert array of 4 Point2Ds: {(0,0),(-1,1),(-1,-1),(1,-1)}
pair stored as: {1,2}
vector of ints: {5,4,2,1}
set of doubles: {1.1,2.2,2.4,3.7}
set of ints: {1,2,3,4,5}
multiset of ints: {1,1,1,1,2,2,2,3,3,4,5}
list of Point2Ds: {(1,2),(3.4,5.4),(0,0)}
valarray of doubles: {1.1,2.2,3.3,4.4,5.5,6.6}
array of Point2D: {(0,0),(-1,1),(-1,-1),(1,-1)}
```

8.11 DeSalvo Standard Library/desalvo/std_cout.h File Reference

Overloads for operator<< for standard library containers.

```
#include <iostream>
#include <valarray>
#include <vector>
#include <list>
#include <algorithm>
#include <set>
#include <string>
#include <initializer_list>
#include <array>
#include <utility>
```

Namespaces

- namespace desalvo_standard_library

  think of this namespace like std or boost, I typically use dsl as an alias.

Functions

- template<typename T , typename F >
  std::ostream & operator<< (std::ostream &out, const std::pair< T, F > &vec)
- template<typename T >
  std::ostream & operator<< (std::ostream &out, const std::vector< T > &vec)
- template<typename T >
  std::ostream & operator<< (std::ostream &out, const std::initializer_list< T > &vec)
- template<typename T >
  std::ostream & operator<< (std::ostream &out, const std::multiset< T > &my_list)
- template<typename T >
  std::ostream & operator<< (std::ostream &out, const std::set< T > &my_list)
- template<typename T >
  std::ostream & operator<< (std::ostream &out, const std::valarray< T > &vec)
- template<typename T , size_t n>
  std::ostream & operator<< (std::ostream &out, const std::array< T, n > &vec)
- template<typename T >
  std::ostream & operator<< (std::ostream &out, const std::slice_array< T > &vec)
- template<typename T >
  std::ostream & operator<< (std::ostream &out, const std::list< T > &my_list)
- template<typename T , typename String = std::string>
  void desalvo_standard_library::print (T &&container, std::string ending="", std::ostream &out=std::cout, String separation=std::string(","), String open_bracket=std::string("{"), String close_bracket=std::string("}"))
- template<typename T , typename String = std::string>
  void **desalvo_standard_library::print** (T &&container, std::string begin_with="{", std::string separate_by=",", std::string end_with="}", std::ostream &out=std::cout)

Variables

- std::string **cout_separation** = ","
- std::string **cout_open_bracket** = "{"
- std::string **cout_close_bracket** = "}"
- std::string **no_ending** = ""

8.11.1 Detailed Description

Overloads for operator<< for standard library containers.

Author

Stephen DeSalvo

Date

December, 2014 This is a collection of overloads to operator<< that allows for printing of collections of objects. The default perferred format is {1,2,3,4,5}, that is, elements separated by commas and the entire list enclosed in { }.

Only include this file if you resign yourself to the exact same style as I prefer. Otherwise consider writing your own or waiting for another version to come out which is a bit more general.

```
// Sample code

// Overloads of operator<< for standard library containers
#include "std_cout.h"

int main(int argc, const char * argv[])
{
```

```
std::vector<double> v {1.2, 3.14, 5.555, -1.234};
dsl::print("vector: ");
dsl::print(v, dsl::start_new_line);

std::list<bool> switches {true, true, false, false, true, false, false, true };
dsl::print("list: ");
dsl::print(switches, dsl::start_new_line);

std::set<int> s;
s.insert(5); s.insert(1); s.insert(-123); s.insert(4);
s.insert(5); s.insert(12); s.insert(7); s.insert(4);
dsl::print("set: ");
dsl::print(s, dsl::start_new_line);

std::multiset<int> multi;
multi.insert(5); multi.insert(1); multi.insert(-123); multi.insert(4);
multi.insert(5); multi.insert(12); multi.insert(7); multi.insert(4);
dsl::print("multiset: ");
dsl::print(multi, dsl::start_new_line);

std::valarray<size_t> vals {1,12,0,4,4,3,2,9,9,7};
dsl::print("valarray: ");
dsl::print(vals, dsl::start_new_line);

std::vector< std::pair<int, double> > vp { {1,.01}, {2,.04}, {3,1.23}, {4,-.0184} };
dsl::print("vector of pairs: ");
dsl::print(vp, dsl::start_new_line);

auto v {1, -1, 23, -756, 222, 5, 4, -3, 77, 18};
dsl::print("default collection: ");
dsl::print(v,dsl::start_new_line);

return 0;
}
```

The output should be:

```
vector: {1.2,3.14,5.555,-1.234}
list: {1,1,0,0,1,0,0,1}
set: {-123,1,4,5,7,12}
multiset: {-123,1,4,4,5,5,7,12}
valarray: {1,12,0,4,4,3,2,9,9,7}
vector of pairs: {{1,0.01},{2,0.04},{3,1.23},{4,-0.0184}}
```

8.11.2 Function Documentation

8.11.2.1 template<typename T , typename F > std::ostream & operator<< (std::ostream & *out*, const std::pair< T, F > & *vec*)

Templated function for output of pair<T,F> format.

Template Parameters

T	is the first element type
F	is the second element type

Parameters

out	is the output stream
vec	is the pair<T,F> to output

Returns

the output stream.

```
#include "desalvo/std_cout.h"

namespace dsl = desalvo_standard_library;

int main(int argc, const char * argv[]) {

std::pair<int, char> p {5, 'c'};
std::pair<double, int> p2 {3.14159, -12};
```

```
std::pair<std::pair<double, int>, char> p3 { {-1.23, 12},{'a'}};

std::cout << p << std::endl;
std::cout << p2 << std::endl;
std::cout << p3 << std::endl;

return 0;
}
```

Should produce output

```
{5,c}
{3.14159,-12}
{{-1.23,12},a}
```

8.11.2.2 template<typename T > std::ostream & operator<< (std::ostream & *out*, const std::vector< T > & *vec*)

Templated function for output of vector<T> format.

Template Parameters

T	is the type of each element in each of the vectors

Parameters

out	is the output stream
vec	is the vector<T> to output

Returns

the output stream.

```
#include "desalvo/std_cout.h"

int main(int argc, const char * argv[]) {

std::vector<int> v {1,3,7};
std::vector<int> v2{2,6,8,9,1};
std::vector<int> v3{1,1,1,-1};
std::vector<int> v4{0,0,-1,1};

std::vector< std::vector<int> > sv {{1,2,3},{4,5,6},{7,8,9}};
std::vector< std::vector<int> > sv2{v,v2,v3,v4};

std::cout << v << std::endl;
std::cout << v2 << std::endl;
std::cout << v3 << std::endl;
std::cout << v4 << std::endl;

std::cout << sv << std::endl;
std::cout << sv2<< std::endl;

return 0;
}
```

Should produce output

```
{1,3,7}
{2,6,8,9,1}
{1,1,1,-1}
{0,0,-1,1}
{{1,2,3},{4,5,6},{7,8,9}}
{{1,3,7},{2,6,8,9,1},{1,1,1,-1},{0,0,-1,1}}
```

8.11.2.3 template<typename T > std::ostream & operator<< (std::ostream & *out*, const std::initializer_list< T > & *my_list*)

Templated function for output of std::list<T> format.

Template Parameters

is	the type of each element

Parameters

out	is the output stream
my_list	is the list<T> to output

Returns

the output stream.

```
#include "desalvo/std_cout.h"

int main(int argc, const char * argv[]) {

auto v  {1,2,3,4,5};
auto v2  {0,0};
auto v3  {-1,2,-1234};

std::initializer_list<std::initializer_list<int>> sv {{1,2},{1,3,5,9},{0,0,-1}};
auto sv2 {v,v2,v3};


std::cout << v << std::endl;
std::cout << v2 << std::endl;
std::cout << v2 << std::endl;

std::cout << sv << std::endl;
std::cout << sv2 << std::endl;

return 0;
}
```

Should produce output

```
{1,2,3,4,5}
{0,0}
{0,0}
{{1,2},{1,3,5,9},{0,0,-1}}
{{1,2,3,4,5},{0,0},{-1,2,-1234}}
```

8.11.2.4 template<typename T > std::ostream & operator<< (std::ostream & *out*, const std::multiset< T > & *my_list*)

Templated function for output of std::multiset<T> format.

Template Parameters

T	is the type of each element

Parameters

out	is the output stream
my_list	is the list<T> to output

Returns

the output stream.

```
#include "desalvo/std_cout.h"

int main(int argc, const char * argv[]) {

std::multiset<int> v  {1,2,3,4,5};
std::multiset<int> v2  {0,0};
std::multiset<int> v3  {-1,2,-1234};

std::cout << v << std::endl;
std::cout << v2 << std::endl;
```

```
std::cout << v3 << std::endl;

return 0;
}
```

Should produce output

```
{1,2,3,4,5}
{0,0}
{-1234,-1,2}
```

8.11.2.5 template<typename T > std::ostream & operator<< (std::ostream & *out*, const std::set< T > & *my_list*)

Templated function for output of std::set<T> format.

Template Parameters

T	is the type of each element

Parameters

out	is the output stream
my_list	is the list<T> to output

Returns

the output stream.

```
#include "desalvo/std_cout.h"

int main(int argc, const char * argv[]) {

std::set<int> v  {1,2,3,4,5};
std::set<int> v2  {0,0};
std::set<int> v3  {-1,2,-1234};

std::cout << v << std::endl;
std::cout << v2 << std::endl;
std::cout << v3 << std::endl;

return 0;
}
```

Should produce output

```
{1,2,3,4,5}
{0}
{-1234,-1,2}
```

8.11.2.6 template<typename T > std::ostream & operator<< (std::ostream & *out*, const std::valarray< T > & *vec*)

Templated function for output of std::valarray<T> format.

Template Parameters

T	is the type of each element in the std::valarray

Parameters

out	is the output stream
vec	is the vector<T> to output

Returns

the output stream.

```
#include "desalvo/std_cout.h"

int main(int argc, const char * argv[]) {

std::valarray<double> v {1.1,2.2,3.3,4.1,5.6,6.3,7.8};
std::valarray<std::pair<int, int>> v2{ {1,2},{2,3},{4,5},{-1,-1},{0,-2}};

std::cout << v << std::endl;
std::cout << v2 << std::endl;

return 0;
}
```

Should produce output

```
{1.1,2.2,3.3,4.1,5.6,6.3,7.8}
{{1,2},{2,3},{4,5},{-1,-1},{0,-2}}
```

8.11.2.7 template<typename T , size_t n> std::ostream & operator<< (std::ostream & *out*, const std::array< T, n > & *vec*)

Templated function for output of std::array<T> format.

Template Parameters

T	is the type of each element in the std::array
n	is the size of the std::array

Parameters

out	is the output stream
vec	is the array<T> to output

Returns

the output stream.

```
int main(int argc, const char * argv[]) {

std::array<int,5> v  {1,2,3,4,5};
std::array<int,2> v2  {0,0};
std::array<int,3> v3  {-1,2,-1234};

std::cout << v << std::endl;
std::cout << v2 << std::endl;
std::cout << v3 << std::endl;

return 0;
}
```

Should produce output

```
{1,2,3,4,5}
{0,0}
{-1,2,-1234}
```

8.11.2.8 template<typename T > std::ostream & operator<< (std::ostream & *out*, const std::slice_array< T > & *vec*)

Templated function for output of std::slice_array<T> format.

Template Parameters

T	is the type of each element

Parameters

out	is the output stream
vec	is the vector<T> to output

Returns

the output stream.

8.11.2.9 template<typename T > std::ostream & operator<< (std::ostream & *out,* const std::list< T > & *my_list*)

Templated function for output of std::list<T> format.

Template Parameters

is	the type of each element

Parameters

out	is the output stream
my_list	is the list<T> to output

Returns

the output stream.

```
#include "desalvo/std_cout.h"

int main(int argc, const char * argv[]) {

std::list<int> v  {1,2,3,4,5};
std::list<int> v2  {0,0};
std::list<int> v3  {-1,2,-1234};

std::cout << v << std::endl;
std::cout << v2 << std::endl;
std::cout << v3 << std::endl;

return 0;
}
```

Should produce output

```
{1,2,3,4,5}
{0,0}
{-1,2,-1234}
```

8.12 DeSalvo Standard Library/desalvo/table.h File Reference

classes pertaining to 2-dimensional tables of values

```
#include <iostream>
#include <memory>
#include <numeric>
#include <initializer_list>
#include <vector>
#include <array>
#include "numerical.h"
```

Classes

- class desalvo_standard_library::table< ValueType, WorkingPrecision >

stores a 2-dimensional table of values

- class desalvo_standard_library::table< ValueType, WorkingPrecision >::const_iterator

  random access const iterator for all entries in table
- class desalvo_standard_library::table< ValueType, WorkingPrecision >::row_const_iterator

  Random Access const_iterator for Rows, muentry.
- class desalvo_standard_library::table< ValueType, WorkingPrecision >::column_const_iterator

  Random Access const_iterator for columns.
- class desalvo_standard_library::table< ValueType, WorkingPrecision >::iterator

  random access iterator for a table, treating it like a 1D array
- class desalvo_standard_library::table< ValueType, WorkingPrecision >::row_iterator

  Random Access Iterator for Rows, muentry.
- class desalvo_standard_library::table< ValueType, WorkingPrecision >::column_iterator

  Random Access Iterator for columns.
- class desalvo_standard_library::table< ValueType, WorkingPrecision >::table_row_reference

  reference to a row of a table, useful for range-based for loops
- class desalvo_standard_library::table< ValueType, WorkingPrecision >::table_column_reference

  reference to a column of a table, useful for range-based for loops, C++11

Namespaces

- namespace desalvo_standard_library

  think of this namespace like std or boost, I typically use dsl as an alias.

Functions

- template<typename ValueType , typename WorkingPrecision >
  bool **desalvo_standard_library::operator!=** (const table< ValueType, WorkingPrecision > &lhs, const table< ValueType, WorkingPrecision > &rhs)

8.12.1 Detailed Description

classes pertaining to 2-dimensional tables of values Provides a base class, dsl::table<ValueType,WorkingPrecision>, which can be inherited from for use with matrices or contingency tables. Provides basic access, and supports C++11-style generic algorithms like range-based for loops.

The dsl::table stores its elements contiguously, so it can be used with many libraries requiring a matrix or table of values stored in a contiguous array.