The Set Library

0.2.1

Generated by Doxygen 1.5.8

Mon Jan 24 01:13:02 2011

Contents

Chapter 1

C Data Structure Library: Set Library

Version:

0.2.1

Author:

Jun Woong (woong.jun at gmail.com)

Date:

last modified on 2011-01-24

1.1 Introduction

This document specifies the Set Library which belongs to the C Data Structure Library. The basic structure is from David Hanson's book, "C Interfaces and Implementations." I modifies the original implementation to make it more appropriate for my other projects and to enhance its readibility; for example a prefix is used more strictly in order to avoid the user namespace pollution.

Since the book explains its design and implementation in a very comprehensive way, not to mention the copyright issues, it is nothing but waste to repeat it here, so I finish this document by giving introduction to the library; how to use the facilities is deeply explained in files that define them.

The Set Library reserves identifiers starting with `set_` and `SET_`, and imports the Assertion Library (which requires the Exception Handling Library) and the Memory Management Library.

1.2 How to Use The Library

The Set Library implements sets whose concept does not differ from that in mathematics. You can also regard it as a table where values do not have associated keys and act as keys by themselves. Since a value in a set is also a key, two instances of the same value refer to the same element in the set. The storage used to maintain a set itself is managed by the library, but any storage allocated for data stored in sets should be managed by a user program.

Similarly for other data structure libraries, use of the Set Library follows this sequence: create, use and destroy. Except for functions to inspect sets, all other functions do one of them in various ways.

set_new() that creates an empty set takes three unusual arguments. The first one is a hint for the expected length of the set it creates, and the other two are to specify user-defined functions that perform creation and comparison of hash values to represent members. Some important conditions that those functions have to satisfy are described in set_new().

In general, a null pointer given to an argument expecting a set is considered an error which results in an assertion failure, but the functions for set operations (set_union(), set_inter(), set_minus() and set_diff()) take a null pointer as a valid argument and treat it as representing an empty set. Also note that they always produce a distinct set; none of them alters the original set.

1.3 Boilerplate Code

Using a set starts with creating one using set_new(). As explained in the function, it is important to provide three arguments properly. If members to a set are generated by the Hash Library, the second and third arguments can be granted null pointers, which lets internal default functions used for the set. There are other ways to create sets from an existing set with set_union(), set_inter(), set_minus() and set_diff() (getting a union, intersection, difference, symmetric difference of sets, respectively); since this library does not define the concept of a universe, no support for a complement. All set creation functions allocates a storage for a set and if no allocation is possible, an exception is raised instead of returning a failure indicator like a null pointer.

Once a set created, a member can be added to and removed from a set using set_put() and set_remove(). Adding a member to a set also entails memory allocation, and thus an exception can be raised. set_member() inspects if a set contains a specific member, and set_length() gives the number of members in a set, a.k.a. the length of a set.

There are two ways to apply some operations on every member in a set; set_map() takes a user-defined function and calls it for each of members, and set_toarray() converts a set into a dynamic arrays. A storage for the generated array is allocated by the library (thus, an exception is possible again), but a user program is responsible for releasing the storage when the array is no longer necessary.

set_free() takes a set and releases the storage used to maintain it. Note that any storage allocated by a user program to contain or represent members is not deallocated by the library.

As an example, the following code creates two sets (whose expected length is set to 20 and members are generated by the Hash Library) and those sets have input characters as members by turns. It then obtains from them a union and an intersection and enumerates members in the resultsing sets.

```
int c;
char b;
unsigned i = 0;
void **pa, **pb;
set_t *myset1, *myset2, *u, *t;

myset1 = set_new(20, NULL, NULL);
myset2 = set_new(20, NULL, NULL);

while ((c = getchar()) != EOF) {
    b = c;
    set_put((i++ % 2 == 0)? myset1: myset2, hash_new(&b, 1));
}

u = set_union(myset1, myset2);
t = set_inter(myset1, myset2);

set_free(&myset1);
set_free(&myset2);

pa = set_toarray(u, NULL);
printf("union: ");
for (pb = pa; *pb; pb++)
    printf("%c", *(char *)*pb);
MEM_FREE(pa);

pa = set_toarray(t, NULL);
printf("\nintersection: ");
for (pb = pa; *pb; pb++)
    printf("%c", *(char *)*pb);
MEM_FREE(pa);

hash_reset();
set_free(&u);
set_free(&t);
```

where hash_new() and hash_reset() come from the Hash Library, and MEM_NEW() and MEM_FREE() from the Memory Management Library.

Things to note include:

- it is not a character itself but a hash value for it that is put into a set;
- since all set operations produce a distinct set, it is possible to destroy `myset1` and `myset2` after their union and intersection obtained;
- arrays generated by set_toarray() have to be deallocated by a user code; and
- storages for members in a set should be released by a user code because set_free() does not care about them (members are hash values given by hash_new() in this example, thus hash_reset() is used to release storages for them.)

1.4 Future Directions

1.4.1 Storing Hash Numbers

Modifying the data structure for sets have hash numbers explicitly makes it possible for a user-provided hashing function to be called only once for each member in sets and for a user-provided comparison function to be called only when the hash numbers differ, which leads to the performance improvement.

1.4.2 Improvement on Set Operations

Set operations like set_union(), set_inter(), set_minus() and set_diff() can be improved when two sets on which the operations are performed have the same number of buckets by applying the operations to each pair of corresponding buckets.

1.5 Contact Me

Visit `http://project.woong.org` to get the lastest version of this library. Only a small portion of my homepage (`http://www.woong.org`) is maintained in English, thus one who is not good at Korean would have difficulty when navigating most of other pages served in Korean. If you think the information you are looking for is on pages written in Korean you cannot read, do not hesitate to send me an email asking for help.

Any comments about the library are welcomed. If you have a proposal or question on the library just email me, and then I will reply as soon as possible.

1.6 Copyright

I do not wholly hold the copyright of this library; it is mostly held by David Hanson as stated in his book, "C: Interfaces and Implementations:"

Copyright (c) 1994,1995,1996,1997 by David R. Hanson.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

For the parts I added or modified, the following applies:

Copyright (C) 2009-2011 by Jun Woong.

This package is a set implementation by Jun Woong. The implementation was written so as to conform with the Standard C published by ISO 9899:1990 and ISO 9899:1999.

Chapter 2

Todo List

Global set_diff Improvements are possible and planned:

- the code can be modified so that the operation is performed on each pair of corresponding buckets when two given sets have the same number of buckets.

Global set_inter Improvements are possible and planned:

- the code can be modified so that the operation is performed on each pair of corresponding buckets when two given sets have the same number of buckets.

Global set_minus Improvements are possible and planned:

- the code can be modified so that the operation is performed on each pair of corresponding buckets when two given sets have the same number of buckets.

Global set_union Improvements are possible and planned:

- the code can be modified so that the operation is performed on each pair of corresponding buckets when two given sets have the same number of buckets.

Chapter 3

File Index

3.1 File List

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

Chapter 4

File Documentation

4.1 set.c File Reference

Source for Set Library (CDSL).

`#include <limits.h>`

`#include <stddef.h>`

`#include "cbl/memory.h"`

`#include "cbl/assert.h"`

`#include "set.h"`

Include dependency graph for set.c:

Data Structures

- struct **set_t**
- struct **set_t::set_t::member**

Defines

- #define MAX(x, y) ((x) > (y)? (x): (y))
- #define MIN(x, y) ((x) > (y)? (y): (x))

Functions

- set_t ∗() set_new (int hint, int cmp(const void ∗, const void ∗), unsigned hash(const void ∗))

  creates a new set.

- int() set_member (set_t ∗set, const void ∗member)

  inspects if a set contains a member.

- void() set_put (set_t ∗set, const void ∗member)

  puts a member to a set.

- void ∗() set_remove (set_t ∗set, const void ∗member)

  removes a member from a set.

- size_t() set_length (set_t ∗set)

  returns the length of a set.

- void() set_free (set_t ∗∗pset)

  destroys a set.

- void() set_map (set_t ∗set, void apply(const void ∗member, void ∗cl), void ∗cl)

  calls a user-provided function for each member in a set.

- void ∗∗() set_toarray (set_t ∗set, void ∗end)

  converts a set to an array.

- set_t ∗() set_union (set_t ∗s, set_t ∗t)

  returns a union set of two sets.

- set_t ∗() set_inter (set_t ∗s, set_t ∗t)

  returns an intersection of two sets.

- set_t ∗() set_minus (set_t ∗s, set_t ∗t)

  returns a difference set of two sets

- set_t ∗() set_diff (set_t ∗s, set_t ∗t)

  returns a symmetric difference of two sets.

4.1.1 Detailed Description

Source for Set Library (CDSL).

4.1.2 Define Documentation

4.1.2.1 #define MAX(x, y) ((x) > (y)? (x): (y))

returns the larger of two

4.1.2.2 #define MIN(x, y) ((x) > (y)? (y): (x))

returns the smaller of two

4.1.3 Function Documentation

4.1.3.1 set_t*() set_diff (set_t * *s*, set_t * *t*)

returns a symmetric difference of two sets.

set_diff() returns a symmetric difference of two sets, that is a set with members only one of two operand sets has. A symmetric difference set is identical to the union set of (`s - t`) and (`t - s`). See set_union() for more detailed explanation commonly applied to the set operation functions.

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: foreign data structure given for `s` or `t`

Parameters:

← ***s*** operand of set difference operation

← ***t*** operand of set difference operation

Returns:

symmetric difference set

Todo

Improvements are possible and planned:

- the code can be modified so that the operation is performed on each pair of corresponding buckets when two given sets have the same number of buckets.

Here is the call graph for this function:

4.1.3.2 void() set_free (set_t ∗∗ *pset*)

destroys a set.

set_free() destroys a set by deallocating the storage for it and set a given pointer to a null pointer. As always, set_free() does not deallocate any storage for members in the set, which must be done by a user.

Possible exceptions: assert_exceptfail

Unchecked errors: pointer to foreign data structure given for `pset`

Parameters:

↔ ***pset*** pointer to set to destroy

Returns:

nothing

4.1.3.3 set_t∗() set_inter (set_t ∗ *s*, set_t ∗ *t*)

returns an intersection of two sets.

set_inter() returns an intersection of two sets, that is a set with only members that both have in common. See set_union() for more detailed explanation commonly applied to the set operation functions.

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: foreign data structure given for `s` or `t`

Parameters:

← ***s*** operand of set intersection operation

← ***t*** operand of set intersection operation

Returns:

intersection set

Todo

Improvements are possible and planned:

- the code can be modified so that the operation is performed on each pair of corresponding buckets when two given sets have the same number of buckets.

Here is the call graph for this function:

Here is the caller graph for this function:

4.1.3.4 size_t() set_length (set_t ∗ *set*)

returns the length of a set.

set_length() returns the length of a set which is the number of all members in a set.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `set`

Parameters:

← ***set*** set whose length returned

Returns:

length of set

4.1.3.5 void() set_map (set_t ∗ *set*, void *apply*const void ∗member, void ∗cl, void ∗ *cl*)

calls a user-provided function for each member in a set.

For each member in a set, set_map() calls a user-provided callback function; it is useful when doing some common task for each member. The pointer given in `cl` is passed to the third parameter of a user callback function, so can be used as a communication channel between the caller of set_map() and the callback. Differently from table_map(), set_map() gives a member (which is a pointer value) to apply() rather than a pointer to a member (which is a pointer to pointer value); compare the type of the `member` parameter of apply() to that of the `value` parameter of apply() in table_map(). This is very natural since a member plays a role of a hashing key in a set as a key does in a table. Also note that apply() is not able to modify a member itself but can do a value pointed by the member.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `set`

Warning:

The order in which a user-provided function is called for each member is unspecified; a set is an unordered collection of members, so this is not a limiting factor, however.

Parameters:

↔ ***set*** set with which `apply` called
← ***apply*** user-provided function (callback)
← ***cl*** passing-by argument to `apply`

Returns:

nothing

4.1.3.6 int() set_member (set_t ∗ *set*, const void ∗ *member*)

inspects if a set contains a member.

set_member() inspects a set to see if it contains a given member.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `set`

Parameters:

← ***set*** set to inspect
← ***member*** member to find in a set

Returns:

found/not found indicator

Return values:

0 member not found
1 member found

Here is the caller graph for this function:

4.1.3.7 set_t∗() set_minus (set_t ∗ *s*, set_t ∗ *t*)

returns a difference set of two sets

set_inter() returns a difference of two sets, that is a set with members that `t` has but `does` not. See set_union() for more detailed explanation commonly applied to the set operation functions.

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: foreign data structure given for `s` or `t`

Parameters:

← ***s*** operand of set difference operation

← ***t*** operand of set difference operation

Returns:

difference set, `s` - `t`

Todo

Improvements are possible and planned:

- the code can be modified so that the operation is performed on each pair of corresponding buckets when two given sets have the same number of buckets.

Here is the call graph for this function:

4.1.3.8 set_t∗() set_new (int *hint*, int *cmp*const void ∗, const void ∗, unsigned *hash*const void ∗)

creates a new set.

set_new() creates a new set. It takes some information on a set it will create:

- `hint`: an estimate for the size of a set;
- `cmp`: a user-provided function for comparing members;
- `hash`: a user-provided function for generating a hash value from a member

set_new() determines the size of the internal hash table kept in a set based on `hint`. It never restricts the number of members one can put into a set, but a better estimate probably bring better performance.

A function given to `cmp` should be defined to take two arguments and to return a value less than, equal to or greater than zero to indicate that the first argument is less than, equal to or greater than the second argument, respectively.

A function given to `hash` takes a member and returns a hash value that is to be used as an index for internal hash table in a set. If the `cmp` function returns zero (which means they are equal) for some two members, the `hash` function must generate the same value for them.

If a null pointer is given for `cmp` or `hash`, the default comparison or hashing function is used; see defhashCmp() and defhashGen(), in which case members are assumed to be hash strings generated by the Hash Library. An example is given below:

```
set_t *myset = set_new(hint, NULL, NULL);
...
set_put(hash_string("member1"));
set_put(hash_string("member2"));
assert(set_member(hash_string("member1")));
```

Possible exceptions: mem_exceptfail

Unchecked errors: invalid functions for `cmp` and `hash`

Parameters:

← ***hint*** hint for size of hash table

← ***cmp*** user-defined comparison function

← ***hash*** user-defined hash generating function

Returns:

new set created

Here is the caller graph for this function:

4.1.3.9 void() set_put (set_t $*$ *set*, const void $*$ *member*)

puts a member to a set.

set_put() inserts a member to a set. If it fails to find a member matched to a given member, it inserts the given member to a set after proper storage allocation. If it finds a matched one, it replaces an existing member with a given one. Even if they compare equal by a user-defined comparison function, they may have different representations. Thus, replacing the old one with the new one makes sense.

Note that a member is a pointer. If members are, say, integers in an application, objects to contain them are necessary to put them into a set.

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: foreign data structure given for `set`

Parameters:

- ↔ ***set*** set to which member inserted
- ← ***member*** member to insert

Returns:

nothing

Here is the caller graph for this function:

4.1.3.10 void∗() set_remove (set_t ∗ *set*, const void ∗ *member*)

removes a member from a set.

set_remove() gets rid of a member from a set. Note that set_remove() does not deallocate any storage for the member to remove, which must be done by a user.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `set`

Parameters:

- ↔ ***set*** set from whcih member removed
- ← ***member*** member to remove

Returns:

previous member or null pointer

Return values:

non-null	previous member
NULL	member not found

Warning:

If the stored member is a null pointer, an ambiguous situation may occur.

4.1.3.11 void∗∗() set_toarray (set_t ∗ *set*, void ∗ *end*)

converts a set to an array.

set_toarray() converts members stored in a set to an array. The last element of the constructed array is assigned by `end`, which is a null pointer in most cases. Do not forget deallocate the array when it is unnecessary.

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: foreign data structure given for `set`

Warning:

> The size of an array generated from an empty set is not zero, since there is always an end-mark.
> As in set_map(), the order in which an array is created for each member is unspecified.

Parameters:

- ← ***set*** set for which array generated
- ← ***end*** end-mark to save in last element of array

Returns:

> array generated from set

4.1.3.12 set_t∗() set_union (set_t ∗ *s*, set_t ∗ *t*)

returns a union set of two sets.

set_union() creates a union set of two given sets and returns it. One of those may be a null pointer, in which case a null pointer is considered an empty set. For every operation, its result constitutes a distinct set from its operand sets, which means the set operations always allocate storage for their results even when one of the operands is empty.

Note that the inferface of the Set Library does not assume the concept of a universe, which is the set of all possible members. Thus, a set operation like the complement of a set is not defined and not implemented.

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: foreign data structure given for `s` or `t`

Warning:

> To create a union of two sets, they have to share the same comparison and hashing functions.

Parameters:

- ← ***s*** operand of set union operation
- ← ***t*** operand of set union operation

Returns:

> union set

Todo

> Improvements are possible and planned:

- the code can be modified so that the operation is performed on each pair of corresponding buckets when two given sets have the same number of buckets.

Here is the call graph for this function:

4.2 set.h File Reference

Documentation for Set Library (CDSL).

`#include <stddef.h>`

Include dependency graph for set.h:

This graph shows which files directly or indirectly include this file:

Typedefs

- typedef struct set_t set_t

  represents a set.

Functions

set creating/destroying functions:

- set_t ∗ **set_new** (int, int(const void ∗, const void ∗), unsigned(const void ∗))
- void set_free (set_t ∗∗)

  destroys a set.

data/information retrieving functions:

- size_t set_length (set_t ∗)

  returns the length of a set.

- int set_member (set_t ∗, const void ∗)

  inspects if a set contains a member.

- void set_put (set_t ∗, const void ∗)

  puts a member to a set.

- void ∗ set_remove (set_t ∗, const void ∗)

  removes a member from a set.

set handling functions:

- void **set_map** (set_t ∗, void(const void ∗, void ∗), void ∗)
- void ∗∗ set_toarray (set_t ∗, void ∗)

  converts a set to an array.

set operation functions:

- set_t ∗ set_union (set_t ∗, set_t ∗)

  returns a union set of two sets.

- set_t ∗ set_inter (set_t ∗, set_t ∗)

  returns an intersection of two sets.

- set_t ∗ set_minus (set_t ∗, set_t ∗)

  returns a difference set of two sets

- set_t ∗ set_diff (set_t ∗, set_t ∗)

  returns a symmetric difference of two sets.

4.2.1 Detailed Description

Documentation for Set Library (CDSL).

Header for Set Library (CDSL).

4.2.2 Function Documentation

4.2.2.1 set_t∗ set_diff (set_t ∗ *s*, set_t ∗ *t*)

returns a symmetric difference of two sets.

set_diff() returns a symmetric difference of two sets, that is a set with members only one of two operand sets has. A symmetric difference set is identical to the union set of (`s - t`) and (`t - s`). See set_union() for more detailed explanation commonly applied to the set operation functions.

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: foreign data structure given for `s` or `t`

Parameters:

← ***s*** operand of set difference operation

← ***t*** operand of set difference operation

Returns:

symmetric difference set

Todo

Improvements are possible and planned:

- the code can be modified so that the operation is performed on each pair of corresponding buckets when two given sets have the same number of buckets.

Here is the call graph for this function:

4.2.2.2 void set_free (set_t ∗∗ *pset*)

destroys a set.

set_free() destroys a set by deallocating the storage for it and set a given pointer to a null pointer. As always, set_free() does not deallocate any storage for members in the set, which must be done by a user.

Possible exceptions: assert_exceptfail

Unchecked errors: pointer to foreign data structure given for `pset`

Parameters:

↔ ***pset*** pointer to set to destroy

Returns:

nothing

4.2.2.3 set_t∗ set_inter (set_t ∗ *s*, set_t ∗ *t*)

returns an intersection of two sets.

set_inter() returns an intersection of two sets, that is a set with only members that both have in common. See set_union() for more detailed explanation commonly applied to the set operation functions.

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: foreign data structure given for `s` or `t`

Parameters:

← ***s*** operand of set intersection operation

← ***t*** operand of set intersection operation

Returns:

intersection set

Todo

Improvements are possible and planned:

- the code can be modified so that the operation is performed on each pair of corresponding buckets when two given sets have the same number of buckets.

Here is the call graph for this function:

Here is the caller graph for this function:

4.2.2.4 size_t set_length (set_t ∗ *set*)

returns the length of a set.

set_length() returns the length of a set which is the number of all members in a set.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `set`

Parameters:

← ***set*** set whose length returned

Returns:

length of set

4.2.2.5 int set_member (set_t * *set*, const void * *member*)

inspects if a set contains a member.

set_member() inspects a set to see if it contains a given member.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `set`

Parameters:

← ***set*** set to inspect

← ***member*** member to find in a set

Returns:

found/not found indicator

Return values:

0 member not found

1 member found

Here is the caller graph for this function:

4.2.2.6 set_t* set_minus (set_t * *s*, set_t * *t*)

returns a difference set of two sets

set_inter() returns a difference of two sets, that is a set with members that `t` has but `does` not. See set_union() for more detailed explanation commonly applied to the set operation functions.

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: foreign data structure given for `s` or `t`

Parameters:

← ***s*** operand of set difference operation

← ***t*** operand of set difference operation

Returns:

difference set, `s - t`

Todo

Improvements are possible and planned:

- the code can be modified so that the operation is performed on each pair of corresponding buckets when two given sets have the same number of buckets.

Here is the call graph for this function:

4.2.2.7 void set_put (set_t ∗ *set*, const void ∗ *member*)

puts a member to a set.

set_put() inserts a member to a set. If it fails to find a member matched to a given member, it inserts the given member to a set after proper storage allocation. If it finds a matched one, it replaces an existing member with a given one. Even if they compare equal by a user-defined comparison function, they may have different representations. Thus, replacing the old one with the new one makes sense.

Note that a member is a pointer. If members are, say, integers in an application, objects to contain them are necessary to put them into a set.

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: foreign data structure given for `set`

Parameters:

↔ ***set*** set to which member inserted

← ***member*** member to insert

Returns:

nothing

Here is the caller graph for this function:

4.2.2.8 void∗ set_remove (set_t ∗ *set*, const void ∗ *member*)

removes a member from a set.

set_remove() gets rid of a member from a set. Note that set_remove() does not deallocate any storage for the member to remove, which must be done by a user.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `set`

Parameters:

↔ ***set*** set from whcih member removed
← ***member*** member to remove

Returns:

previous member or null pointer

Return values:

non-null previous member
NULL member not found

Warning:

If the stored member is a null pointer, an ambiguous situation may occur.

4.2.2.9 void∗∗ set_toarray (set_t ∗ *set*, void ∗ *end*)

converts a set to an array.

set_toarray() converts members stored in a set to an array. The last element of the constructed array is assigned by `end`, which is a null pointer in most cases. Do not forget deallocate the array when it is unnecessary.

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: foreign data structure given for `set`

Warning:

The size of an array generated from an empty set is not zero, since there is always an end-mark.
As in set_map(), the order in which an array is created for each member is unspecified.

Parameters:

← ***set*** set for which array generated
← ***end*** end-mark to save in last element of array

Returns:

array generated from set

4.2.2.10 set_t∗ set_union (set_t ∗ *s*, set_t ∗ *t*)

returns a union set of two sets.

set_union() creates a union set of two given sets and returns it. One of those may be a null pointer, in which case a null pointer is considered an empty set. For every operation, its result constitutes a distinct set from its operand sets, which means the set operations always allocate storage for their results even when one of the operands is empty.

Note that the inferface of the Set Library does not assume the concept of a universe, which is the set of all possible members. Thus, a set operation like the complement of a set is not defined and not implemented.

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: foreign data structure given for `s` or `t`

Warning:

To create a union of two sets, they have to share the same comparison and hashing functions.

Parameters:

← ***s*** operand of set union operation

← ***t*** operand of set union operation

Returns:

union set

Todo

Improvements are possible and planned:

- the code can be modified so that the operation is performed on each pair of corresponding buckets when two given sets have the same number of buckets.

Here is the call graph for this function:

Index