set<Key, Compare, Alloc>

Containers

Type

Category: containers

Component type: type

Description

`set` is a Sorted Associative Container that stores objects of type `Key`. `set` is a Simple Associative Container, meaning that its value type, as well as its key type, is `Key`. It is also a Unique Associative Container, meaning that no two elements are the same.

`set` and `multiset` are particularly well suited to the set algorithms `includes`, `set_union`, `set_intersection`, `set_difference`, and `set_symmetric_difference`. The reason for this is twofold. First, the set algorithms require their arguments to be sorted ranges, and, since `set` and `multiset` are Sorted Associative Containers, their elements are always sorted in ascending order. Second, the output range of these algorithms is always sorted, and inserting a sorted range into a `set` or `multiset` is a fast operation: the Unique Sorted Associative Container and Multiple Sorted Associative Container requirements guarantee that inserting a range takes only linear time if the range is already sorted.

`set` has the important property that inserting a new element into a `set` does not invalidate iterators that point to existing elements. Erasing an element from a set also does not invalidate any iterators, except, of course, for iterators that actually point to the element that is being erased.

Example

```
struct ltstr
{
  bool operator()(const char* s1, const char* s2) const
  {
    return strcmp(s1, s2) < 0;
  }
};

int main()
{
  const int N = 6;
  const char* a[N] = {"isomer", "ephemeral", "prosaic",
                      "nugatory", "artichoke", "serif"};
  const char* b[N] = {"flat", "this", "artichoke",
                      "frigate", "prosaic", "isomer"};

  set<const char*, ltstr> A(a, a + N);
  set<const char*, ltstr> B(b, b + N);
  set<const char*, ltstr> C;

  cout << "Set A: ";
  copy(A.begin(), A.end(), ostream_iterator<const char*>(cout, " "));
  cout << endl;
  cout << "Set B: ";
  copy(B.begin(), B.end(), ostream_iterator<const char*>(cout, " "));
  cout << endl;

  cout << "Union: ";
  set_union(A.begin(), A.end(), B.begin(), B.end(),
            ostream_iterator<const char*>(cout, " "),
            ltstr());
  cout << endl;

  cout << "Intersection: ";
  set_intersection(A.begin(), A.end(), B.begin(), B.end(),
                   ostream_iterator<const char*>(cout, " "),
                   ltstr());
  cout << endl;

  set_difference(A.begin(), A.end(), B.begin(), B.end(),
                 inserter(C, C.begin()),
                 ltstr());
  cout << "Set C (difference of A and B): ";
  copy(C.begin(), C.end(), ostream_iterator<const char*>(cout, " "));
  cout << endl;
}
```

Definition

Defined in the standard header set, and in the nonstandard backward-compatibility header set.h.

Template parameters

Parameter	Description	Default
`Key`	The set's key type and value type. This is also defined as `set::key_type` and `set::value_type`	
`Compare`	The key comparison function, a Strict Weak Ordering whose argument type is `key_type`; it returns `true` if its first argument is less than its second argument, and `false` otherwise. This is also defined as `set::key_compare` and `set::value_compare`.	`less<Key>`
`Alloc`	The `set`'s allocator, used for all internal memory management.	`alloc`

Model of

Unique Sorted Associative Container, Simple Associative Container

Type requirements

- `Key` is Assignable.
- `Compare` is a Strict Weak Ordering whose argument type is `Key`.
- `Alloc` is an Allocator.

Public base classes

None.

Members

Member	Where defined	Description
`value_type`	Container	The type of object, `T`, stored in the set.
`key_type`	Associative Container	The key type associated with `value_type`.
`key_compare`	Sorted Associative Container	Function object that compares two keys for ordering.
`value_compare`	Sorted Associative Container	Function object that compares two values for ordering.
`pointer`	Container	Pointer to `T`.
`reference`	Container	Reference to `T`
`const_reference`	Container	Const reference to `T`
`size_type`	Container	An unsigned integral type.
`difference_type`	Container	A signed integral type.
`iterator`	Container	Iterator used to iterate through a `set`.
`const_iterator`	Container	Const iterator used to iterate through a `set`. (`Iterator` and `const_iterator` are the same type.)
`reverse_iterator`	Reversible Container	Iterator used to iterate backwards through a `set`.
`const_reverse_iterator`	Reversible Container	Const iterator used to iterate backwards through a `set`. (`Reverse_iterator` and `const_reverse_iterator` are the same type.)
`iterator begin() const`	Container	Returns an `iterator` pointing to the beginning of the `set`.
`iterator end() const`	Container	Returns an `iterator` pointing to the end of the `set`.
`reverse_iterator rbegin() const`	Reversible Container	Returns a `reverse_iterator` pointing to the beginning of the reversed set.
`reverse_iterator rend() const`	Reversible Container	Returns a `reverse_iterator` pointing to the end of the reversed set.
`size_type size() const`	Container	Returns the size of the `set`.
`size_type max_size() const`	Container	Returns the largest possible size of the `set`.
`bool empty() const`	Container	`true` if the `set`'s size is `0`.

`key_compare key_comp() const`	Sorted Associative Container	Returns the `key_compare` object used by the `set`.
`value_compare value_comp() const`	Sorted Associative Container	Returns the `value_compare` object used by the `set`.
`set()`	Container	Creates an empty `set`.
`set(const key_compare& comp)`	Sorted Associative Container	Creates an empty `set`, using `comp` as the `key_compare` object.

`template <class InputIterator>`
`set(InputIterator f, InputIterator l)`
[1]
	Unique Sorted Associative Container	Creates a set with a copy of a range.

`template <class InputIterator>`
`set(InputIterator f, InputIterator l,`
`    const key_compare& comp)`
[1]
	Unique Sorted Associative Container	Creates a set with a copy of a range, using `comp` as the `key_compare` object.

`set(const set&)`	Container	The copy constructor.
`set& operator=(const set&)`	Container	The assignment operator
`void swap(set&)`	Container	Swaps the contents of two sets.

`pair<iterator, bool>`
`insert(const value_type& x)`
	Unique Associative Container	Inserts `x` into the `set`.

`iterator insert(iterator pos,`
`                const value_type& x)`
	Unique Sorted Associative Container	Inserts `x` into the `set`, using `pos` as a hint to where it will be inserted.

`template <class InputIterator>`
`void insert(InputIterator, InputIterator)`
[1]
	Unique Sorted Associative Container	Inserts a range into the `set`.

`void erase(iterator pos)`	Associative Container	Erases the element pointed to by `pos`.
`size_type erase(const key_type& k)`	Associative Container	Erases the element whose key is `k`.
`void erase(iterator first, iterator last)`	Associative Container	Erases all elements in a range.
`void clear()`	Associative Container	Erases all of the elements.
`iterator find(const key_type& k) const`	Associative Container	Finds an element whose key is `k`.
`size_type count(const key_type& k) const`	Unique Associative Container	Counts the number of elements whose key is `k`.

`iterator lower_bound(const key_type& k) const`	Sorted Associative Container	Finds the first element whose key is not less than `k`.
`iterator upper_bound(const key_type& k) const`	Sorted Associative Container	Finds the first element whose key greater than `k`.

`pair<iterator, iterator>`
`equal_range(const key_type& k) const`
	Sorted Associative Container	Finds a range containing all elements whose key is `k`.

`bool operator==(const set&,`
`const set&)`
	Forward Container	Tests two sets for equality. This is a global function, not a member function.

`bool operator<(const set&,`
`const set&)`
	Forward Container	Lexicographical comparison. This is a global function, not a member function.

New members

All of `set`'s members are defined in the Unique Sorted Associative Container and Simple Associative Container requirements. `Set` does not introduce any new members.

Notes

[1] This member function relies on *member template* functions, which at present (early 1998) are not supported by all compilers. If your compiler supports member templates, you can call this function with any type of input iterator. If your compiler does not yet support member templates, though, then the arguments must either be of type `const value_type*` or of type `set::const_iterator`.

See also

Associative Container, Sorted Associative Container, Simple Associative Container, Unique Sorted Associative Container, `map`, `multiset`, `multimap`, `hash_set`, `hash_map`, `hash_multiset`, `hash_multimap`