Container	Operation		Short Description
	Category	Function-like Macros Prototype	
CCXLL	Create	*struct* ccxll (TYPE) CCXLL;	Create a ccxll container of type TYPE. Use a C struct to simulate a list container.
		struct ccxll_iter (TYPE) ITER;	Create an iterator for the ccxll container of type TYPE. The iterator might be invalid if the container is modified.
		struct ccxll_pckd (TYPE) CCXLL;	Create a packed ccxll container of type TYPE. This decreases memory usage as much as possible.
		struct ccxll_iter_pckd (TYPE) ITER;	Create an iterator for packed ccxll containers of type TYPE. Packed ccxll containers should select packed iterators to make sure the correctness.
	Initialize	void ccxll_init (CCXLL);	Initialize the ccxll container. CAUTION: Every container should be initialized right after its creation.
		void ccxll_iter_init (ITER, CCXLL);	Initialize the iterator for the ccxll container. CAUTION: Every iterator should be initialized right after its creation.
	Destroy	*stat* ccxll_free (CCXLL);	Deallocate all elements in the container manually. CAUTION: Every container should be destroyed before the program terminates.
	Access	TYPE& ccxll_front (CCXLL);	Return a reference to the first element. It's an undefined behavior if the container is empty.
		TYPE& ccxll_back (CCXLL);	Return a reference to the last element. It's an undefined behavior if the container is empty.
	Capacity	int ccxll_size (CCXLL);	Return the number of the elements in the container. Return 0 if the container is empty.
		int ccxll_empty (CCXLL);	Check whether the container is empty. Return 1 if the container is empty, and return 0 if it is not.
	Modifiers	*stat* ccxll_push_front (CCXLL, **value**);	Insert an element at the beginning. This makes a copy of **value** into the container.
		stat ccxll_push_back (CCXLL, **value**);	Insert an element at the end. This makes a copy of **value** into the container.
		stat ccxll_pop_front (CCXLL);	Remove the first element. There is nothing modified if the container is empty.
		stat ccxll_pop_back (CCXLL);	Remove the last element. There is nothing modified if the container is empty.
		stat ccxll_insert (ITER, **value**);	Insert an element at the position where the iterator points. This makes a copy of **value** into the container.
		stat ccxll_erase (ITER);	Erase an element at the position where the iterator points. There is nothing modified if the container is empty.
		stat ccxll_swap (CCXLL_A, CCXLL_B);	Swap two containers of the same type. It may cause unexpected errors if two containers are of different types.
		stat ccxll_clear (CCXLL);	Remove all elements in the container. This does not deallocate all elements in the container.
	Operations	*stat* ccxll_move_range (ITER_P, ITER_L, ITER_R);	Move the elements in the range [ITER_L, ITER_R) to position where ITER_P points. These three iterators should be affiliated to the same ccxll container.
		stat ccxll_merge_range (ITER_L, ITER_M, ITER_R, ITER_X);	Merge two sorted segments in the range [ITER_L, ITER_M) and [ITER_M, ITER_R). This requires an auxiliary iterator ITER_X which is not in use currently.
		stat ccxll_sort (CCXLL);	Sort all elements with default comparator CCXLL_LEQ_COMPAR. See ccxll_sort_extd for supporting user-defined comparators.
		stat ccxll_reverse_range (ITER_L, ITER_R);	Reverse the elements in the range [ITER_L, ITER_R]. This performs in constant time no matter how large the range is.
	Comparators	int CCXLL_LEQ_COMPAR|XLEQ (ITER_L, ITER_R);	Compare values by passing and dereferencing two iterators for sorting algorithms. Return 1 if the value pointed by ITER_L is not greater than the value pointed by ITER_R.
	Iterators	TYPE& ccxll_iter_dref (ITER);	Return a reference to the element. It's an undefined behavior if the iterator is not invalid.
		TYPE& ccxll_iter_dref_[prev|next] (ITER);	Return a reference to the previous/next element. It's an undefined behavior if the iterator is not invalid.
		void ccxll_iter_copy (ITER_DST, ITER_SRC);	Copy the iterator from ITER_SRC to ITER_DST. It's not acceptable to assign the iterator by assignment operator.
		void ccxll_iter_[head|tail] (ITER, CCXLL);	Set the iterator to the head/tail of the container. The head/tail of the container is the sentinel node pointing to the first/last element.
		void ccxll_iter_[begin|end] (ITER, CCXLL);	Set the iterator to the first/last element usaully. Set the iterator to the tail/head if the container is empty.
		void ccxll_iter_at_[head|tail] (ITER);	Check whether the iterator points to the head/tail of the container. Return 1 if it is true. Otherwise, return 0.
		void ccxll_iter_at_[begin|end] (ITER, CCXLL);	Check whether the iterator points to the first/last element. Return 1 if it is true. Otherwise, return 0.
		void* ccxll_iter_[incr|decr] (ITER);	Move the iterator forward/backward by one element. Return NULL iff the iterator doesn't point to any element before and after moving.
		stat ccxll_iter_advance (ITER, **diff**);	Move the iterator by **diff** element(s). (regard forward as positive) The iterator will stop at the sentinel node if there is no element left to iterate over.
	Traversal	*loop* CCXLL_TRAV (CCXLL) *stat*;	Traverse all elements from the beginning to the end. The built-in .iter is used. This is equilavent to CCXLL_FORWARD_TRAVERSAL (CCXLL, CCXLL.ITER).
		loop CCXLL_[FOR|BACK]WARD_TRAVERSAL (CCXLL, ITER) *stat*;	Traverse all elements from the beginning/end to the end/beginning. There is no effect if the container is empty.
	Extensions	*stat* ccxll_copy (CCXLL_DST, CCXLL_SRC);	Copy the container from CCXLL_SRC to CCXLL_DST. This performs a deep copy of all elements in CCXLL_SRC.
		stat ccxll_rearrange (CCXLL);	Rearrange all elements in the container. This will put adjacent nodes in the nearby memory space for higher cache hit rate.