

Container	Operation		Short Description
	Category	Function-like Macros Prototype	
CCXLL	Create	<u>CCXLL</u> ccxll (TYPE) list;	Create a ccxll container list of type TYPE with an iterator. This is implemented by a C struct to construct a list container.
		<u>CCXLL</u> ccxll_pckd (TYPE) list;	Create a packed ccxll container list of type TYPE with an iterator. This is implemented by an aligned C struct to construct a list container.
		<u>CCXLL</u> ccxll_extd (TYPE , unsigned <i>num</i> , ALIGN) list;	Create a ccxll container list of type TYPE with <i>num</i> iterators. The container is packed when ALIGN is PACKED. Otherwise, set NORMAL for default.
	Initialize	void ccxll_init (CCXLL);	Initialize the ccxll container. Every container must be initialized right after its creation.
		void ccxll_iter_init (ITER, CCXLL);	Initialize the iterator for the ccxll container. Every iterator is implicitly initialized when the container it belongs to is initialized.
	Destroy	<u>stat</u> ccxll_free (CCXLL);	Deallocate all elements in the container manually. Every container should be destroyed before the program terminates.
	Access	TYPE& ccxll_[front back] (CCXLL);	Return a reference to the first/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	<u>stat</u> ccxll_push_[front back] (CCXLL, TYPE <i>value</i>);	Insert an element at the beginning/end. This makes a copy of <i>value</i> into the container.
		<u>stat</u> ccxll_pop_[front back] (CCXLL);	Remove the first/last element. There is nothing modified if the container is empty.
		<u>stat</u> ccxll_insert (ITER, TYPE <i>value</i>);	Insert an element at the position where the iterator points. This makes a copy of <i>value</i> into the container.
		<u>stat</u> ccxll_erase (ITER);	Erase an element at the position where the iterator points. There is nothing modified if the container is empty.
		<u>stat</u> ccxll_swap (CCXLL <i>a</i> , CCXLL <i>b</i>);	Swap two containers of the same type. It may cause unexpected errors if two containers are of different types.
		<u>stat</u> ccxll_resize (CCXLL, int <i>num</i> , TYPE <i>value</i>);	Resize the container to contain <i>num</i> elements. If the current size is smaller than <i>num</i> elements, then fills with <i>value</i> . Otherwise, it truncates.
		<u>stat</u> ccxll_clear (CCXLL);	Remove all elements in the container. This does not deallocate all elements in the container.
	Operations	<u>stat</u> ccxll_move_range (ITER <i>pos</i> , ITER <i>left</i> , ITER <i>right</i>);	Move the elements in the range [<i>left</i> , <i>right</i>) to position where <i>pos</i> points. These three iterators should be affiliated to the same ccxll container.
		<u>stat</u> ccxll_merge[_extd] (CCXLL <i>dst</i> , CCXLL <i>src</i> [, (*LEQ)()]);	Merge two sorted lists from <i>src</i> into <i>dst</i> . Merge with the default comparator XLEQ if _extd postfix is not specified.
		<u>stat</u> ccxll_sort[_extd] (CCXLL[, (*LEQ)()]);	Sort all elements in the list. Sort with the default comparator XLEQ if _extd postfix is not specified.
		<u>stat</u> ccxll_reverse_range (ITER <i>left</i> , ITER <i>right</i>);	Reverse the elements in the range [<i>left</i> , <i>right</i>]. This performs in constant time no matter how large the range is.
	Comparators	LEQ CCXLL_LEQ_COMPAR (ITER <i>a</i> , ITER <i>b</i>); (abbrev. XLEQ)	Compare values by passing and dereferencing two iterators for sorting algorithms. Return 1 iff the value pointed by <i>a</i> is not greater than the value pointed by <i>b</i> . Otherwise, return 0.
	Iterators	ITER ITER[_NTH] (CCXLL [, unsigned <i>num</i>]);	Return the <i>num</i> -th iterator of the list. Return the zero-th iterator if _NTH postfix is not specified.
		TYPE& DREF (ITER);	Return a reference to the element. It's an undefined behavior if the iterator is not (semi-)valid.
		TYPE& DREF_[PREV NEXT] (ITER);	Return a reference to the previous/next element. It's an undefined behavior if the iterator is not valid.
		void ccxll_iter_copy (ITER <i>dst</i> , ITER <i>src</i>);	Copy the iterator from <i>src</i> to <i>dst</i> . It's not acceptable to assign the iterator by assignment operator.
		void ccxll_iter_[head tail] (ITER);	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);	Set the iterator to the first/last element usually. Set the iterator to the tail/head if the container is empty.
		int 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.
		int ccxll_iter_at_[begin end] (ITER);	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.
		<u>stat</u> ccxll_iter_advance (ITER, int <i>diff</i>);	Move the iterator by <i>diff</i> element(s). (regard forward as positive) The iterator will stop at the sentinel node if there is no element left to iterate over.
		<u>stat</u> ccxll_iter_distance (ITER <i>a</i> , ITER <i>b</i> , int * <i>dist</i>);	Return the distance between <i>a</i> and <i>b</i> through <i>dist</i> . Return 0 if the distance between them cannot be determined.
	Traversal	<u>loop</u> CCXLL_[INCR DECR] (ITER) <u>stat</u> ;	Traverse all elements forward/backward. This is implemented by a single for statement.
		<u>loop</u> CCXLL_[INCR DECR]_DREF (TYPE * <i>pval</i> , CCXLL) <u>stat</u> ;	Traverse all elements forward/backward, and set the address of each element into <i>pval</i> . This macro will not be activated if CCC_STRICT is defined.