SSLL Documentation

Class Description – Holds a list of type T objects or primitives in a linked list of nodes that each contain a pointer to the next node.

Node Description – A private container that will hold values of type T and a pointer to the next node in the list.

Node() – Construct a default node holding no data with a null pointer.

~Node() – Properly destroys Node object so as not to leak memory.

Node(const T& data) – Construct a node with an element "data" and a null pointer.

Node(const T& data, const Node* next) – Construct a node with an element "data" and a pointer to the "next" node.

SSLL_Iter Description – Iterates through the list that it belongs to and returns values in the list from start to end. The values in the list that it returns can be manipulated since they are not constant and they are passed by reference.

explicit SSLL_Iter(Node* start = NULL) : here(start) – Explicit Constructor. Unless the start node is explicitly specified, it returns an iterator pointing to the end of the list (by containing a NULL pointer).

SSLL_Iter(const SSLL_Iter& src) : here(src.here) – Copy constructor that sets the new iterator's pointer to the same node in the list as the original iterator's pointer.

reference operator*() const – Dereference operator is overloaded to return the item in the list that the iterator is pointing to unless the iterator is out of range (one past the last element), in which case it will throw an out of range exception.

pointer operator->() const – Member access operator for structs and classes. Similarly to the above method, it will return the item in the list that the iterator is pointing to unless the iterator is out of range, in which case it will throw an out of range exception. Since it returns a pointer to the item in the list, its individual members and properties can be accessed on the right-hand side of the operator.

self_reference operator=(const SSLL_Iter& src) – Overloaded Equals operator so as to return a proper copy construction of the iterator, or itself if it is set to equal itself.

self_reference operator++() – Pre-increment overloaded to step the iterator one element up in the list. Pre-increment will continue to return the list until it is one past the last element. Once it is one past the last element and the user attempts to use the operator again, an out of range exception will be thrown since it is already at the maximum length.

self_type operator++(int) – Post-increment operator is overloaded similarly to the pre-increment, but another iterator is copied before this iterator points to the next element, and that copied iterator is returned for use. Post-increment will continue to return the list until it is one past the last element. Once it is one past the last element and the user attempts to use the operator again, an out of range exception will be thrown since it is already at the maximum length.

bool operator==(const SSLL_Iter& rhs) const – Comparison operator is overloaded to return true IFF two SSLL iterators are pointing to the same element in the same list.

bool operator!=(const SSLL_Iter& rhs) const – Comparison operator is overloaded to return true IFF two SSLL operators are pointing to different elements.

SSLL_Const_Iter Description – Iterates through the list that it belongs to and returns values in the list from start to end. The values in the list that it returns cannot be manipulated since they are constant.

explicit SSLL_Const_Iter(Node* start = NULL) : here(start) – Explicit Constructor. Unless the start node is explicitly specified, it returns an iterator pointing to the end of the list (by containing a NULL pointer).

SSLL_Const_Iter(const SSLL_Const_Iter& src) : here(src.here) – Copy constructor that sets the new iterator's pointer to the same node in the list as the original iterator's pointer.

reference operator*() const – Dereference operator. It will return the item in the list that the iterator is pointing to unless the iterator is out of range, in which case it will throw an out of range exception. The referenced item cannot be manipulated because it is constant.

pointer operator->() const – Member access operator for structs and classes. Similarly to the above method, it will return the item in the list that the iterator is pointing to unless the iterator is out of range, in which case it will throw an out of range exception. Since it returns a pointer to the item in the list, its individual members and properties can be accessed on the right-hand side of the operator. Any class or struct in this list cannot be changed to a new one since it is const.

self_reference operator=(const SSLL_Const_Iter& src) – Overloaded Equals operator so as to return a proper copy construction of the constant iterator, or itself if it is set to equal itself.

self_reference operator++() – Pre-increment overloaded to step the iterator one element up in the list. Pre-increment will continue to return the list until it is one past the last element. Once it is one past the last element and the user attempts to use the operator again, an out of range exception will be thrown since it is already at the maximum length.

self_type operator++(int) – Post-increment operator is overloaded similarly to the pre-increment, but another iterator is copied before this iterator points to the next element, and that copied iterator is returned for use. Post-increment will continue to return the list until it is one

past the last element. Once it is one past the last element and the user attempts to use the operator again, an out of range exception will be thrown since it is already at the maximum length.

bool operator==(const SSLL_Iter& rhs) const – Comparison operator is overloaded to return true IFF two const SSLL iterators are pointing to the same element in the same list.

bool operator!=(const SSLL_Iter& rhs) const – Comparison operator is overloaded to return true IFF two const SSLL operators are pointing to different elements.

SSLL() – Default constructor for the SSLL. It sets the head and tail pointers to NULL and the size to zero because there are zero elements in the list.

SSLL(const SSLL& copy) – Copy constructor for the SSLL. It is a deep copy constructor in that it recreates the nodes with the values from all of the original nodes, so the original list is not copied by reference. Thus, if the original list is altered after a new list is copied, the copied list will remain the same.

~SSLL() – Deconstructor for SSLL. Destroys all nodes in the chain by calling clear().

SSLL& operator=(const SSLL& src) – Equals operator overloaded so that it will return the same object if it is being "reassigned" to the same object. Otherwise it will copy the list that it is newly assigned to and destroy itself.

T& operator[](int pos) – The subscript operator is overloaded so that each element in the list can be retrieved by putting its position in the list in the subscript. Ideally, it will work for any number 0 through n-1, where n is the number of items in the list. If any number outside of this range is entered, a domain error will get thrown informing the user that it is an invalid location.

T const& operator[](int pos) – This const subscript operator overload is exactly the same as the above operator, but this one is called when the list is declared as const. Thus, items that are returned using this operator cannot be changed, and are not passed by reference.

T replace(const T& ele, int pos) – Replaces the element at a particular position in the list with a different element. If the position is not in the bounds of 0 to n-1, where n is the amount of elements in the list, a domain error is thrown, and no element is replaced.

@param ele: The element that is put in as a replacement.
@param pos: The position in the list at which the element is to be replaced.

@returns the object that was replaced in the list.

void insert(const T& ele, int pos) – Inserts an element at position i in the list. Everything in the list at position i and after position i gets shifted right one in the list. If position i is one past the last position, it gets inserted as the tail. If position i > n where n is the size of the list or i < 0, a domain error exception is thrown.

@param ele: the element to be inserted into the list
@param pos: the position in the list where the element is to be inserted

void push_back(const T& ele) – Adds an element to the back of the list in a new node with a null pointer.

@param ele: The element to be added to the back of the list

void push_front(const T& ele) – Adds an element to the front of the list, pushing all positions one position back in the list.

@param ele: The element to be inserted at the front of the list

T remove(int pos) – Removes an element at a specified position, causing all elements after it to "shift" one left in the list, which means the node that was pointing to what was removed points to the next node in the list. If the position specified is not within the scope of the list, where -1 < scope < n elements, a domain error is thrown.

@param pos: The position in the list to remove and return the element.

@returns the element that was removed

T pop_back() – Pops the element on the end of the list off of the list. If there is nothing in the list to return, a length error is thrown.

@returns the element that was at the end of the list.

T pop_front() – Pops off the element at the front of the list and returns it. If there is nothing in the list to return, a length error is thrown.

@returns the element that was removed from the list.

T item_at(int pos) – Returns the element at a specified position in the list without altering its position or the list itself. If pos < 0 or pos >= n where n is the number of elements in the list, a domain error is thrown and nothing is returned.

@param pos: The position in the list where the desired element is located.

@returns the element at the specified position in the list.

bool is_empty() – Returns true IFF the list does not contain a single element.

size_t size() – Returns the current size of the entire list; in other words, how many elements are currently contained in the list.

void clear() – Deletes all of the nodes an cleans up the linked list so that there are no elements, and all of the memory required to maintain the list is freed up.

bool contains(const T& ele, bool equals(const T& a, const T& b)) const – Compares every element in the list to a specified element using a specified function until a match is found, at which point "true" will be returned. False will be returned if every element in the list is compared to the specified element and there are no matches.

> @param ele: The element that is to be compared to those in the list
> @param equals(const T& a, const T& b): A function passed in to compare two elements.
>
> @returns true IFF the list contains the parameter "ele".

std::ostream& print(std::ostream& out) const – Takes an ostream and uses that to print out all of the data in the list, one line by one line.

> @param out: The ostream with which the list is printed out.
>
> @returns the osteam used to print the list out.

iterator begin() – Returns an SSLL_Iter that points to the first node in the list. The elements accessed by this iterator can be changed since the list is not constant. This begin() is called because the list is not constant.

iterator end() – Returns an SSLL_Iter that points to one past the last element in the list. This end() is called because the list is not constant.

const_iterator begin() const – Returns an SSLL_Const_Iter that points to the first element in the list. The elements accessed by this iterator cannot be changed since the list is constant. This begin() is called because the list is constant.

const_iterator end() const – Returns an SSLL_Const_Iter that points to one past the last element. This end() is called because the list is constant.