Pool, Singly Linked List

PSLL()
Constructor for the PSLL. The constructor initializes the private variables of the PSLL to NULL and the count to zero

PSLL(const PSLL& src)
Copy constructor.

~PSLL()
Default destructor for the class

T replace(const T &e, int position)
This method takes an element of type T and a position and replaces the node at given position with a new one that is taken from the free list if the free list is not empty. If the free list is empty then a new code is allocated. If the user tries to pass a position that is not valid, then an out of range exception is thrown. Positions have to range within position 0 and the max size of the list.

void PSLL<T>::insert(const T &e, int position)
This method takes an element of type T and a position and inserts a node at the given position. This inserted node is taken from the free list is the free list is not empty. If not then it simply allocates a new node as it did in the simply singly linked list. The method inserts the node at this given position and provides a link between this node and the one that used to be in its place so that the chain is not broken. The method only accepts positions that are within position 0 and the size of the list. This means that an insert can occur one after the last element, just as if the user was push_back(). If the position does not fall within this range, then an out of range exception is thrown.

void PSLL<T>::push_front(const T &e)
This method allocates a new node and pushes it at the beginning of the list. The new node is taken from the free list is the free list is not empty. If the list is empty, then it simply allocates a new node and places it as the first item in the list.

void PSLL<T>::push_back(const T &e)
This method allocates a new node and places it at the back of the list. The new node is taken from the free list is the free list is not empty. Just as the push front method, if no element is yet in the list, then this method creates a new node and places it in the list.

T PSLL<T>::pop_front()
This method removes the head of the list and returns the value that was stored in the head. The node that is popped is added to the free list. It then proceeds to make the new head of the list, the node that proceeded the previous head. If the list is empty, then a logic error is thrown telling the user that the list is empty and the first element in the list cannot be popped.

T PSLL<T>::pop_back()
This method removes the last element from the list and returns the value that was stored in it. The node that is popped is added to the free list. Just as pop_front(), this method also throws a logic error if the user tries to pop from an empty list.

T PSLL<T>::remove(int position)
This method takes an integer position and removes the item at that position. It also returns the value that was stored in that node. The node that is removed from the list is added to the free list. It links the chain of nodes back together. If the list is empty when the remove method is called, then a logic error will be thrown at the user. If the given integer position is not within 0 and the list's size, then an out of range exception will be thrown to the user.

T PSLL<T>::itemAt(int position)
This method takes an integer position and returns the data of the element at that position. Just like the other methods that take an integer position, if the given position is not within the boundaries of the list, then an out of range exception is thrown telling the user that the position they entered was invalid.

bool PSLL<T>::isEmpty() const
This method checks if the list is empty. It returns true if the list is empty and false otherwise.

int PSLL<T>::size() const
This method returns the current size of the list.

void PSLL<T>::clear()
This method empties out the list.

bool PSLL<T>::contains(T e, bool equals_function)
This method checks to see if a given element value is in the list. It returns true if it is and false if it is not.

const ostream& PSLL<T>::print(ostream& out)
This method prints out the list. It stores the result in the ostream.

PSLL<T>::PSLL_ITER PSLL<T>::begin()
This method returns an iterator to the beginning of the list

PSLL<T>::PSLL_ITER PSLL<T>::end()
This method returns an iterator to the end of the list.

PSLL<T>::PSLL_Const_ITER PSLL<T>::begin() const
This method returns a constant iterator to the beginning of a constant list.

PSLL<T>::PSLL_Const_ITER PSLL<T>::end() const
This method returns a constant iterator to the end of a constant list.

T& operator[](int i)
This operator overloading method lets the list be indexed like an array even though it is a linked list.

T const& operator[](int i)const
This operator overloading method lets the constant list be indexed like an array even though it is a linked list.

PSLL_ITER
explicit PSLL_ITER(node *n)
Explicit constructor

PSLL_ITER(const PSLL_ITER& src)
Copy constructor

pointer operator->()
Returns the data stored in what the iterator is pointing at.

reference operator *()
Returns the data stored in what the iterator is pointing at.

self_reference operator++()
Pre-increment operator overloading.

self_type operator++(int)
post-increment operator overloading

bool operator ==(PSLL_ITER &rhs)
Equality operator overloading. Compares to see if two iterators are the same.

bool operator!=(PSLL_ITER &rhs)
Inequality operator overload. Compares to see if two iterators are not the same.

self_reference operator=(PSLL_ITER &rhs)
Assignment operator.

ostream &operator<<(ostream &obj)
Allows for iterators to print the data that they are pointing at.

PSLL_Const_ITER
The following are the methods in the PSLL_Const_ITER class. Their descriptions are the same as the preceding class except that these overloads are for a constant iterator.

explicit PSLL__ConstITER(node *n)
PSLL_ITER(const PSLL_Const_ITER& src)
pointer operator->()
reference operator *()
self_reference operator++()
self_type operator++(int)
ool operator ==(PSLL_Const_ITER &rhs)
bool operator!=(PSLL_Const-ITER &rhs)
self_reference operator=(PSLL_Const_ITER &rhs)
ostream &operator<<(ostream &obj)