Simple Dynamic Array-Based List
SDAL

SDAL()
Constructor for the SDAL. The constructor takes no parameters and simply initializes the size of the array to 50, allocates space for a new dynamic array and sets the amount of numbers in the array(the count) to zero.

SDAL(int arrSize)
This constructor creates takes integer value and allocates a new dynamic array with the given size. It also sets the count of items in that array to zero.

SDAL(const SDAL& src)
Copy constructor.

~SDAL()
Default destructor for the class

void SDAL<T>::Reduce()
Private method that reduces the array's size by half. It allocates a new array with the given arrSize and copies the elements over to it.

bool SDAL<T>::isFull()
Private method that checks to see if the array is full.

T replace(const T &e, int position)
This method takes an element of type T and a position and replaces the value at given position with a new one. 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 SDAL<T>::insert(const T &e, int position)
This method takes an element of type T and a position and inserts a value at the given position. 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. If the array is full at the time of insertion then it allocates a new array of 1.5 times the size of the one being used and copies the elements over. Then it proceeds with the insertion.

void SDAL<T>::push_front(const T &e)
This method moves all of the elements one index forward and inserts an element at the beginning of the list. If the list is full at the time this method is called, then it will copy the array into an array that is 1.5 times the size of the original and insert at position 0 in the new array.

void SDAL<T>::push_back(const T &e)
This method places an element at the end of the list. If the list is full at the time of pushing back, then a new array 1.5 times the size of the current one will be created and the elements will be copied over to that array. Once that is done, then push_back() will place an element at the end of the list.

T SDAL<T>::pop_front()
This method removes the element at the beginning of the list and returns the value that was stored in it. It then shifts all of the elements down one index to accommodate the removal of the first element. If a user tries to pop from an empty list, then a logic error will be thrown at them, telling them that they cannot pop from an empty list. If once the element is removed from the front, the array's size is greater than 100 and amount of elements in the array are less than half of that, then the array will get reduced by half when the reduce() method is called from within this method.

T SDAL<T>::pop_back()
This method removes the last element from the list and returns the value that was stored in it. Just as pop_front(), this method also throws a logic error if the user tries to pop from an empty list. If once the element is removed from the back, the array's total size is greater than 100 and the amount of elements in the array are less than half of that, then the array will get reduced by half when the reduce() method is called from within this method.

T SDAL<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 slot. It then shifts the elements down one slot. 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. If once the element is removed, the array's size is greater than 100 and amount of elements in the array are less than half of that, then the array will get reduced by half when the reduce() method is called from within this method

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

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

int SDAL<T>::sizeofArray() const
This method is mostly used for testing purposes. It lets the user know the max size of the current array. The preceding method, size, returns the amount of items in the list but not the max amount of items that can be stored in the given list.

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

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

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

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

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

SDAL<T>::SDAL_Const_ITER SDAL<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.

T const& operator[](int i)const
This operator overloading method lets the constant list be indexed like an array.

SDAL_ITER
explicit SDAL_ITER(T* l)
Explicit constructor which takes a pointer to an element. It sets the private variable ptr of the class equal to that given pointer.

SDAL_ITER(const SDAL_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 ==(SDAL_ITER &rhs)
Equality operator overloading. Compares to see if two iterators are the same.

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

self_reference operator=(SDAL_ITER &rhs)
Assignment operator.

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

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

explicit SDAL_Const_ITER(node *n)
SDAL_ITER(const SDAL_Const_ITER& src)
pointer operator->()
reference operator *()
self_reference operator++()
self_type operator++(int)
ool operator ==(SDAL_Const_ITER &rhs)
bool operator!=(SDAL_Const-ITER &rhs)
self_reference operator=(SDAL_Const_ITER &rhs)
ostream &operator<<(ostream &obj)