Wenlan Tian
COP 3530
Section 1087, MAEB 211
11/25/2014
Project 1

Generate informal documentation

Chained Dynamic Array-based List (CDAL)

The template class CDAL is the main class for this part of the project. All the other classes, functions are under this class.

1. The class contains a private Node, which has a *next* pointer pointing to the next node and a *arr* pointer points to a array. The initial node has the *next* pointer pointing to NULL and *arr* pointer pointing to a array of size 50.

2. The class has head pointer pointing to the first nodes, while a tail pointer pointing to the end of the slot in the arrays.

3. The class has *mySize* variable. mySize is used to track the number of the current items.

4. The class has a constructor CDAL() and destructor ~CDAL(). The constructor will initialize the list to an empty list that mySize to 0, head to a new empty node and tial to the first slot in the array.

5. The class overload the copy operator= which will copy a list to the current list. If the copied list is empty, it will throw an exception; if they are they copied list and current list is the same, do nothing; otherwise, safely dispose the current list's content, copy the nodes to current list one by one through push_back() function.

6. The class has several member functions.
(1) T replace(const T& element, int position) will replace the existing element at the specified position with the specified element and returns the original element. If the list is empty, it will throw an out_of_range exception; if the replace position is not within the current list range, it will throw a domain_error exception.

(2) void insert(const T& element, int position) will add the specified element to the list at the specified position, shifting the element originally at that and those in subsequent positions one position to the ”right“. If the insertion position is not within the current list range, it will throw a domain_error exception; if the insertion position is 0, it will call the push_front(element) function; if the insertion position is the end of the list, it will call the push_back(element) function.

(3) void push_front(const T& element) will prepend the specified element to the list.

(4) void push_back(const T& element) will append the specified element to the list.

(5) T pop_front() removes and returns the element at the list's head. If current list empty, it throws an out_of_range exception.

(6) T pop_back() removes and returns the element at the list's tail. If current list empty, it throws an out_of_range exception.

(7) T remove(int position) removes and returns the the element at the specified position, shifting the subsequent elements one position to the ”left“. If current list empty, it will throw a out_of_range exception; if the remove position is not within the current list range, it will throw a domain_error exception.

(8) T item_at(int position) const returns (without removing from the list) the element at the specified position. If the current list is empty, it will throw a out_of_range exception; if the item position is not within the current list range, it will throw a domain_error exception.

(9) bool is_empty() const returns true IFF the list contains no elements.

(10) size_t size() const returns the number of elements in the list.

(11) void clear() removes all elements from the list safely with the pop_back() function.

(12) bool contains(const T& element, bool equals(const T& a, const T& b)) const returns true IFF one of the elements of the list matches the specified element. If the current list empty, it will throw an out_of_range exception.

(13) std::ostream& print(std::ostream& out) const. If the list is empty, it will print<empty list>"; otherwise, if will print out all the elments in the list.

(14) T& operator[](int i) returns a reference to the indexed element. If the current list empty, it will throw an out_of_range exception; if the index i is not within the range of current list, it will throw a domain_error exception.

(15) T const& operator[](int i) const returns an immutable reference to the indexed element. If the current list empty, it will throw an out_of_range exception; if the index i is not within the range of current list, it will throw a domain_error exception.

(16) shrink() will deallocate half the unused arrays whenever the more than half of the arrays are unused (they would all be at the end of the chain).

7. The class defines several types, size_t size_t, T value_type, CDAL_Iter iterator and CDAL_Const_Iter const_iterator.

8. The class has two sub classes CDAL_Iter and CDAL_Const_Iter. These two classes are for iterators. They have several member functions as following:

(1) begin() will construct and return an iterator denoting the list's first element.

(2) end() will construct and return an iterator denoting one past the list's last element.

(3) reference operator*() const returns the item value

(4) pointer operator->() const returns the item value

(5) self_reference operator=(const CDAL_Const_Iter& src) copy the the iterator src to the current iterator.

(6) self_reference operator++() return the value before increment.

(7) self_type operator++(int) return the value after increment.

(8) bool operator==(const CDAL_Const_Iter& rhs) const tests if two iterators equal.

(9) bool operator!=(const CDAL_Const_Iter& rhs) const test if two iterators not equal.