COP4530 – Data Structures, Algorithms and Generic Programming
Recitation 4
Date: 16th September, 2011

Lab topic:

1) **Discus Quiz 2**
2) **Discussion on Assignment 2**
3) **Take Quiz 3**

Discussion on Assignment 2.

Your task is to **write 3 template classes** and ***exercise recursion technique*** and ***analyze runtime*** .

Part 1: Vector Template Class (Functionalities)

a. `Vector ():`

i. The **`size`** is initialized to 0 since we do not have any elements in a newly declared vector.

ii. The **`capacity`** is initialized to 2 since the project requirement states that the default constructor "*initializes an array of size 2*".

b. `~Vector():`

i. Deallocate the dynamically allocated memory for the array.

ii. Deallocate any other dynamically allocated memory

iii. Set **`size`** and **`capacity`** to 0.

c. `T& operator [] (int ind):`

i. Check the bounds for the index **`ind`** that is passed in. If the index is invalid, print out an error message.

ii. If the index is valid, return the value of the element located at the index **`ind`** of the array.

d. `int size() const:`

i. Returns the size of the array.

e. `int capacity() const:`

i. Returns the capacity of the array.

f. `void push_back(const T&):`

i. Add element at the end. Check to see if there is currently enough space to add T. If there is, just add T to the array

ii. If there isn't enough space, reallocate memory for a larger array. You may do so by **a factor of 4** the capacity of the array. Copy the contents over to the new larger array and then add T to the array.

g. `void push_front(const T&):`

i. Add elements in the front. Check to see if there is currently enough space to add T. If there is, just add T to the array

ii. If there isn't enough space, reallocate memory for a larger array. You may do so by **a factor of 4** the capacity of the array. Copy the contents over to the new larger array and then add T to the array.

h. `void pop_back():`

i. Removes elements from the end of the vector.

ii. Reduce the size correspondingly.

i. `void pop_front():`

i. Removes elements from the front of the vector.

ii. Shift elements forward.

iii. Reduce the size correspondingly.

j. `void dump(std::ostream &os ) const:`

i. Prints out the contents of the array.

ii. This method is optional.

If you do not implement **copy constructor** and **assignment operator** then you must declare them private to avoid accidental misuse.

Part 2: A *generic* template `Stack` class

1. Your task is to build a *generic* template **`Stack`** class. This class will contain an instance of the aforementioned **`Vector`** class for storage.

2. The stack class must have the following features (with conventional meaning):

   a) `void push(T &)`
   b) `void pop()`
   c) `T& top()`
   d) `bool empty()`

3. The amortized time complexities for all operations should be O(1). How to do that ?

Part 3: A *generic* template `Queue` class

1. Your task is to build a *generic* template **`Queue`** class.

2. The queue class must have the following features (with conventional meaning):

   a) `void push(T &)`

b) `void pop()`
c) `T& front()`
d) `bool empty()`

3. The amortized time complexities for all operations should be O(1). Any ideas how to do it ?

   Hint: You can use a circular array.

Part 4:Recursion

Provide a recursive implementation to the function mentioned in the assignment.

Merge sort is a popular sorting algorithm which works as follows:

a) Say the input array is of size ***n***
b) If the array contain a single element , then simply return that element. Otherwise divide the array in the middle to produce two lists of size n/2
c) Recursively call Merge sort function on these two sub-arrays
d) Merge the two lists returned by merge sort (assume that merge takes ***cn*** time, where **c** is a constant > 0)
e) Return the merged list.

Basically these steps can be written as:

$$f(n) = 2 * f(n/2) + c * n$$

Calculate the complexity of this algorithm.

References

Topic	**Links**
STL `vector`	**1.** http://www.sgi.com/tech/stl/Vector.html
STL `stack`	1. http://www.cplusplus.com/reference/stl/stack/
STL `queue`	1. http://www.cplusplus.com/reference/stl/queue/