**STL’s Assignment**

1. Create a map and show usage of below functions ->

* begin() – Returns an iterator to the first element in the map
* end() – Returns an iterator to the theoretical element that follows last element in the map
* size() – Returns the number of elements in the map
* max\_size() – Returns the maximum number of elements that the map can hold
* empty() – Returns whether the map is empty
* pair insert(keyvalue, mapvalue) – Adds a new element to the map
* erase(iterator position) – Removes the element at the position pointed by the iterator
* erase(const g)– Removes the key value ‘g’ from the map
* clear() – Removes all the elements from the map

1. Create a vector and show usage of below functions ->

* begin() – Returns an iterator pointing to the first element in the vector
* end() – Returns an iterator pointing to the theoretical element that follows the last element in the vector
* rbegin() – Returns a reverse iterator pointing to the last element in the vector (reverse beginning). It moves from last to first element
* rend() – Returns a reverse iterator pointing to the theoretical element preceding the first element in the vector (considered as reverse end)
* cbegin() – Returns a constant iterator pointing to the first element in the vector.
* cend() – Returns a constant iterator pointing to the theoretical element that follows the last element in the vector.
* crbegin() – Returns a constant reverse iterator pointing to the last element in the vector (reverse beginning). It moves from last to first element
* crend() – Returns a constant reverse iterator pointing to the theoretical element preceding the first element in the vector (considered as reverse end)

1. Define a class that will hold the set of integers from 0 to 31. An element can be set with the set member function and cleared with the clear member function. It is not an error to set an element that's already set or clear an element that's already clear. The function test is used to tell whether an element is set.

Member functions:

void small\_set::set(int item); // Set an element in the set

void small\_set::clear(int item); // Clear an element in the set

int small\_set::test(void); // See whether an element is set

Sample usage:

small\_set a\_set;

a\_set.set(3); // Set contains [3]

a\_set.set(5); // Set contains [3,5]

a\_set.set(5); // Legal (set contains [3,5])

cout << a\_set.test(3) << '\n'; // Prints "1"

cout << a\_set.test(0) << '\n'; // Prints "0"

a\_set.clear(5); // Set contains [3]

1. I have a simple method of learning foreign vocabulary words. I write the words down on a list of flash cards. I then go through the stack of flash cards one at a time. If I get a word right, that card is discarded. If I get it wrong, the card goes to the back of the stack.

Write a class to implement this system.

Member functions:

struct single\_card {

char question[40]; // English version of the word

char answer[40]; // Other language version of the word

};

// Constructor -- takes a list of cards to

// initialize the flash card stack

void flash\_card::flash\_card(single\_card list[]);

// Get the next card

const single\_card &flash\_card::get\_card(void);

//The student got the current card right

void flash\_card::right(void);

// The student got the current card wrong

void flash\_card::wrong(void);

//Returns 1 -- done / 0 -- more to do

int done(void);