I**mpl**ementing Data Structures (4 Problems)

# **Problem1: Building Linked lists**

# **Summary:**

This program develops a **linked list class** like the one provided in the C++ STL. The **public interface** provides a template class of basic functions that’s your linked list supports any data type. In addition, it has an **iterator class** **as an inner class** in order to **access** **the data** stored in the list.

# **Example: list<int> myList; myList.push\_back(1); myList.push\_back(2); myList.push\_back(3); list<int>::iterator it = myList.begin(); it++; cout<< \*it;** // The usage of the scope operator in the declaration of the iterator is because the iterator class is defined as an inner class inside the list class. **List Public Interface:**

The list class has the following public interface:   
• **list()** – default constructor.   
• **list(type value, int initial\_size)**   
• **~list()** – a destructor to clear the list and leave no memory leaks.   
• **int size()** – returns the current number of elements in the list.   
• **void insert(type value, iterator position)** – adds an element at position specified by the iterator.   
• **iterator erase(iterator position)** – erases the element specified by the iterator and return an iterator to the next element, throws exception if position points **after the last element, dummy node**.   
• **list<type>& operator = (list<type> another\_list)** – overloads the assignment operator to **deep copy** a list into another list and return the current list by reference.   
• **iterator begin()** – returns an iterator pointing to **the first element**.   
• **iterator end()** – returns an iterator pointing **after the last element**.

**Iterator Class:**• **void operator ++ ()** – overloads the operator ++, advances the iterator one position towards the end of the list, throws exception if it is currently pointing **after the last element** as STL .end() do.   
• **void operator -- ()** – overloads the operator --, moves the iterator one position toward the beginning of the list, throws exception if it is currently pointing to the first element of the list.   
• **type& operator \* ()** – overloads the **dereference operator** to return the value this node  
• **bool operator == (const iterator &)** – overloads the equality comparison operator, should return true if **the** **passed operator points to the same node**.

# The problem is built using **C++, Visual Studio**.

# **Problem2: Stacks**

# **Summary:**

This program develops a **Stack class** like the one provided in the C++ STL by using the array list implementation as an underlying data structure. The **stack class** is a template class.

**Stack Public Interface:**   
• **stack()** – default constructor.  
• **stack(type value, int intial\_size)**   
• **~stack()** – a destructor to clear the stack and leave no memory leaks.   
• **type& top()** – returns the top element by reference.   
• **void pop()** – removes the top element.   
• **void push(type value)** – adds an element to the top of the stack.   
• **int size()** – returns the number of elements in the stack.

# The problem is built using **C++, Visual Studio**.

# **Problem3: Queues**

# **Summary:**

This program develops a **Queue class** like the one provided in the C++ STL by using the array list implementation as an underlying data structure. The Queue **class** is a template class.

# **Queue Public Interface:** •**queue()** – default constructor. • **queue(type value, int intial\_size)** – constructs a queue having ‘initial\_size’ elements whose values are of **type ‘value’**. • **~queue()** – a destructor to clear the queue and leave no memory leaks. • **type& front()** – returns the first element by reference. • **void pop()** – removes the first element. • **void push(type value)** – adds an element to the back of the queue. • **int size()** – returns the number of elements in the queue.

# The problem is built using **C++, Visual Studio**.

# **Problem4: Stacks using STL Queue**

# **Summary:**

This program develops a **Queue** byusing C++ STL queue as an underlying data structure.   
It provides the only following functions:  
• **int top()** – returns the top element.   
• **void pop()** – removes the top element.   
• **void push(int value)** – adds an element to the top of the stack.

# The problem is built using **C++, Visual Studio**.