1. Implement methods**add, size**in template class **DLinkedList (which implements List ADT)**representing the doubly linked list with type T with the initialized frame. The description of each method is given in the code.

template <class T>  
class DLinkedList {  
public:  
    class Node; // Forward declaration  
protected:  
    Node\* head;  
    Node\* tail;  
    int count;  
public:  
    DLinkedList();  
    ~DLinkedList();  
 void add(const T &e);  
 void add(int index, const T &e);  
 int size();  
public:

class Node

{

private:

T data;

Node \*next;

Node \*previous;

friend class DLinkedList<T>;

public:

Node()

{

this->previous = NULL;

this->next = NULL;

}

Node(const T &data)

{

this->data = data;

this->previous = NULL;

this->next = NULL;

}

};

};

In this exercise, we have include <iostream>, <string>, <sstream> and using namespace std.

**For example:**

| **Test** | **Result** |
| --- | --- |
| DLinkedList<int> list;  int size = 10;  for(int idx=0; idx < size; idx++){  list.add(idx);  }  cout << list.toString(); | [0,1,2,3,4,5,6,7,8,9] |
| DLinkedList<int> list;  int size = 10;  for(int idx=0; idx < size; idx++){  list.add(0, idx);  }  cout << list.toString(); | [9,8,7,6,5,4,3,2,1,0] |

|  |
| --- |
| template <class T>  void DLinkedList<T>::add(const T& e) {  /\* Insert an element into the end of the list. \*/  Node\* newNode = new Node(e);  if (head == NULL){  head = newNode;  tail = newNode;  count++;  }  else{  tail->next = newNode;  newNode->previous = tail;  tail = tail->next;  count++;  }  }  template<class T>  void DLinkedList<T>::add(int index, const T& e) {  /\* Insert an element into the list at given index. \*/  if (index < 0 || index > count)  throw std::out\_of\_range("Out of range");  Node\* newNode = new Node(e);  if (head == NULL){  head = newNode;  tail = newNode;  count++;  return;  }  Node\* cur = head;  Node\* pre = NULL;  for (int i = 0; i < index; i++){  pre = cur;  cur = cur->next;  }  if (pre == NULL){  newNode->next = cur;  cur->previous = newNode;  head = newNode;  }  else{  newNode->next = cur;  pre->next = newNode;  newNode->previous = pre;  if (cur == NULL)  tail = newNode;  else  cur->previous = newNode;  }  count++;  }  template<class T>  int DLinkedList<T>::size() {  /\* Return the length (size) of list \*/  return this->count;  } |

|  | **Test** | **Expected** | **Got** |
| --- | --- | --- | --- |
|  | DLinkedList<int> list;  int size = 10;  for(int idx=0; idx < size; idx++){  list.add(idx);  }  cout << list.toString(); | [0,1,2,3,4,5,6,7,8,9] | [0,1,2,3,4,5,6,7,8,9] |  |
|  | DLinkedList<int> list;  int size = 10;  for(int idx=0; idx < size; idx++){  list.add(0, idx);  }  cout << list.toString(); | [9,8,7,6,5,4,3,2,1,0] | [9,8,7,6,5,4,3,2,1,0] |  |

Passed all tests!

2. Implement methods**get, set, empty, indexOf, contains**in template class D**LinkedList (which implements List ADT)**representing the singly linked list with type T with the initialized frame. The description of each method is given in the code.

template <class T>  
class DLinkedList {  
public:  
    class Node; // Forward declaration  
protected:  
    Node\* head;  
    Node\* tail;  
    int count;  
public:  
    DLinkedList();  
    ~DLinkedList();  
 void add(const T &e);  
 void add(int index, const T &e);  
 int size();  
 bool empty();

T get(int index);

void set(int index, const T &e);

int indexOf(const T &item);

bool contains(const T &item);

public:

class Node

{

private:

T data;

Node \*next;

Node \*previous;

friend class DLinkedList<T>;

public:

Node()

{

this->previous = NULL;

this->next = NULL;

}

Node(const T &data)

{

this->data = data;

this->previous = NULL;

this->next = NULL;

}

};

};

In this exercise, we have include <iostream>, <string>, <sstream> and using namespace std.

**For example:**

| **Test** | **Result** |
| --- | --- |
| DLinkedList<int> list;  int size = 10;  for(int idx=0; idx < size; idx++){  list.add(idx);  }  for(int idx=0; idx < size; idx++){  cout << list.get(idx) << " |";  } | 0 |1 |2 |3 |4 |5 |6 |7 |8 |9 | |
| DLinkedList<int> list;  int size = 10;  int value[] = {2,5,6,3,67,332,43,1,0,9};  for(int idx=0; idx < size; idx++){  list.add(idx);  }  for(int idx=0; idx < size; idx++){  list.set(idx, value[idx]);  }  cout << list.toString(); | [2,5,6,3,67,332,43,1,0,9] |

|  |
| --- |
| template<class T>  T DLinkedList<T>::get(int index) {  /\* Give the data of the element at given index in the list. \*/  Node\* travel = this->head;  if(travel==NULL) return travel->data;  if(index==0) return travel->data;  int place=0;  while(travel->next!=NULL && place!=index)  {  place++;  travel=travel->next;  }  return travel->data;  }  template <class T>  void DLinkedList<T>::set(int index, const T& e) {  /\* Assign new value for element at given index in the list \*/  Node\* travel = this->head;  if(travel==NULL) return;  if(index==0) {travel->data=e; return;}  int place=0;  while(travel->next!=NULL && place!=index)  {  place++;  travel=travel->next;  }  travel->data=e;  return;  }  template<class T>  bool DLinkedList<T>::empty() {  /\* Check if the list is empty or not. \*/  Node\* travel = this->head;  if(travel==NULL) return true;  return false;    }  template<class T>  int DLinkedList<T>::indexOf(const T& item) {  /\* Return the first index wheter item appears in list, otherwise return -1 \*/  Node\* travel = this->head;  if(travel==NULL) return -1;  if(travel->data==item) return 0;  int place=0;  while(travel!=NULL && travel->data!=item)  {  place++;  travel=travel->next;  }  if(travel==NULL) return -1;    return place;  }  template<class T>  bool DLinkedList<T>::contains(const T& item) {  /\* Check if item appears in the list \*/  Node\* travel = this->head;  if(travel==NULL) return false;  if(travel->data==item) return true;  while(travel!=NULL && travel->data!=item)  {  travel=travel->next;  }  if(travel!=NULL) return true;  return false;  } |

| **Test** | **Expected** | **Got** |
| --- | --- | --- |
| DLinkedList<int> list;  int size = 10;  for(int idx=0; idx < size; idx++){  list.add(idx);  }  for(int idx=0; idx < size; idx++){  cout << list.get(idx) << " |";  } | 0 |1 |2 |3 |4 |5 |6 |7 |8 |9 | | 0 |1 |2 |3 |4 |5 |6 |7 |8 |9 | |
| DLinkedList<int> list;  int size = 10;  int value[] = {2,5,6,3,67,332,43,1,0,9};  for(int idx=0; idx < size; idx++){  list.add(idx);  }  for(int idx=0; idx < size; idx++){  list.set(idx, value[idx]);  }  cout << list.toString(); | [2,5,6,3,67,332,43,1,0,9] | [2,5,6,3,67,332,43,1,0,9] |

Passed all tests!

3. Implement methods**removeAt, removeItem, clear**in template class **SLinkedList (which implements List ADT)**representing the singly linked list with type T with the initialized frame. The description of each method is given in the code.

template <class T>  
class DLinkedList {  
public:  
    class Node; // Forward declaration  
protected:  
    Node\* head;  
    Node\* tail;  
    int count;  
public:  
    DLinkedList();  
    ~DLinkedList();  
 void add(const T &e);  
 void add(int index, const T &e);  
 int size();  
 bool empty();

T get(int index);

void set(int index, const T &e);

int indexOf(const T &item);

bool contains(const T &item);

T removeAt(int index);

bool removeItem(const T &item);

void clear();

public:

class Node

{

private:

T data;

Node \*next;

Node \*previous;

friend class DLinkedList<T>;

public:

Node()

{

this->previous = NULL;

this->next = NULL;

}

Node(const T &data)

{

this->data = data;

this->previous = NULL;

this->next = NULL;

}

};

};

In this exercise, we have include <iostream>, <string>, <sstream> and using namespace std.

**For example:**

| **Test** | **Result** |
| --- | --- |
| DLinkedList<int> list;  int size = 10;  int value[] = {2,5,6,3,67,332,43,1,0,9};  for(int idx=0; idx < size; idx++){  list.add(value[idx]);  }  list.removeAt(0);  cout << list.toString(); | [5,6,3,67,332,43,1,0,9] |

|  |
| --- |
| template <class T>  T DLinkedList<T>::removeAt(int index)  {  /\* Remove element at index and return removed value \*/  if (index < 0 || index >= count )  throw std::out\_of\_range("Out of range");  T val;  if (index == 0){  if (count == 1){  val = head->data;  delete head;  head = NULL;  tail = NULL;  }  else {  head->next->previous = NULL;  val = head->data;  Node\* temp = head;  head = head->next;  delete temp;  }  }  else {  Node\* pre = NULL;  Node\* cur = head;  for (int i = 0; i < index; i++){  pre = cur;  cur = cur->next;  }  val = cur->data;  pre->next = cur->next;  if (cur->next == NULL)  tail = pre;  else  cur->next->previous = pre;  delete cur;  }  count--;  return val;  }  template <class T>  bool DLinkedList<T>::removeItem(const T& item)  {  /\* Remove the first apperance of item in list and return true, otherwise return false \*/  int index = indexOf(item);  if (index != -1) {  removeAt(index);  return true;  }  else return false;  }  template<class T>  void DLinkedList<T>::clear(){  /\* Remove all elements in list \*/  while (head != NULL) {  Node\* temp = head;  head = head->next;  delete temp;  count--;  }  tail = NULL;  } |

|  | **Test** | **Expected** | **Got** |
| --- | --- | --- | --- |
|  | DLinkedList<int> list;  int size = 10;  int value[] = {2,5,6,3,67,332,43,1,0,9};  for(int idx=0; idx < size; idx++){  list.add(value[idx]);  }  list.removeAt(0);  cout << list.toString(); | [5,6,3,67,332,43,1,0,9] | [5,6,3,67,332,43,1,0,9] |  |

Passed all tests!

4. Implement all methods in class **Stack** with template type **T**. The description of each method is written as comment in frame code.

#ifndef STACK\_H

#define STACK\_H  
#include "DLinkedList.h"  
template<class T>  
class Stack {  
protected:  
    DLinkedList<T> list;  
public:  
    Stack() {}  
    void push(T item) ;  
    T pop() ;  
    T top() ;  
    bool empty() ;  
    int size() ;  
    void clear() ;  
};  
  
#endif

You can use all methods in class **DLinkedList**without implementing them again. The description of class **DLinkedList**is written as comment in frame code.

template <class T>  
class DLinkedList   
{  
public:  
    class Node;     //forward declaration  
protected:  
    Node\* head;  
    Node\* tail;  
    int count;  
public:  
    DLinkedList() ;  
    ~DLinkedList();  
    void add(const T& e);  
    void add(int index, const T& e);  
    T removeAt(int index);  
    bool removeItem(const T& removeItem);  
    bool empty();  
    int size();  
    void clear();  
    T get(int index);  
    void set(int index, const T& e);  
    int indexOf(const T& item);  
    bool contains(const T& item);  
};

**For example:**

| **Test** | **Result** |
| --- | --- |
| Stack<int> stack;  cout << stack.empty() << " " << stack.size(); | 1 0 |
| Stack<int> stack;  int item[] = { 3, 1, 4, 5, 2, 8, 10, 12 };  for (int idx = 0; idx < 8; idx++) stack.push(item[idx]);    assert(stack.top() == 12);    stack.pop();  stack.pop();    cout << stack.top(); | 8 |

|  |
| --- |
| void push(T item) {  // TODO: Push new element into the top of the stack  DLinkedList<T> p = this->list;  this->list.add(item);  }  T pop() {  // TODO: Remove an element on top of the stack  int size = this->list.size()-1;    return this->list.removeAt(size);  }  T top() {  // TODO: Get value of the element on top of the stack  int size = this->list.size()-1;  T solution = this->list.get(size);  return solution;  }  bool empty() {  // TODO: Determine if the stack is empty  return this->list.empty();  }  int size() {  // TODO: Get the size of the stack  int size = this->list.size();  return size;  }  void clear() {  // TODO: Clear all elements of the stack  this->list.clear();  } |

|  | **Test** | **Expected** | **Got** |
| --- | --- | --- | --- |
|  | Stack<int> stack;  cout << stack.empty() << " " << stack.size(); | 1 0 | 1 0 |  |
|  | Stack<int> stack;  int item[] = { 3, 1, 4, 5, 2, 8, 10, 12 };  for (int idx = 0; idx < 8; idx++) stack.push(item[idx]);  assert(stack.top() == 12);  stack.pop();  stack.pop();  cout << stack.top(); | 8 | 8 |  |

Passed all tests!

5. Implement all methods in class **Queue** with template type **T**. The description of each method is written as comment in frame code.

#ifndef QUEUE\_H

#define QUEUE\_H  
#include "DLinkedList.h"  
template<class T>  
class Queue {  
protected:  
    DLinkedList<T> list;  
public:  
    Queue() {}  
    void push(T item) ;  
    T pop() ;  
    T top() ;  
    bool empty() ;  
    int size() ;  
    void clear() ;  
};  
  
#endif /\* QUEUE\_H \*/

You can use all methods in class **DLinkedList**without implementing them again. The description of class **DLinkedList**is written as comment in frame code.

template <class T>  
class DLinkedList   
{  
public:  
    class Node;     //forward declaration  
protected:  
    Node\* head;  
    Node\* tail;  
    int count;  
public:  
    DLinkedList() ;  
    ~DLinkedList();  
    void add(const T& e);  
    void add(int index, const T& e);  
    T removeAt(int index);  
    bool removeItem(const T& removeItem);  
    bool empty();  
    int size();  
    void clear();  
    T get(int index);  
    void set(int index, const T& e);  
    int indexOf(const T& item);  
    bool contains(const T& item);  
};

**For example:**

| **Test** | **Result** |
| --- | --- |
| Queue<int> queue;  assert(queue.empty());  assert(queue.size() == 0); |  |

|  |
| --- |
| void push(T item) {  // TODO: Push new element into the end of the queue  this->list.add(item);  }  T pop() {  // TODO: Remove an element in the head of the queue  return this->list.removeAt(0);  }  T top() {  // TODO: Get value of the element in the head of the queue  return this->list.get(0);  }  bool empty() {  // TODO: Determine if the queue is empty  return this->list.empty();  }  int size() {  // TODO: Get the size of the queue  return this->list.size();  }  void clear() {  // TODO: Clear all elements of the queue  this->list.clear();  } |

|  | **Test** |
| --- | --- |
|  | |

Passed all tests!

6. Implement method bubbleSort() in class SLinkedList to sort this list in ascending order. After each bubble, we will print out a list to check (using printList).

#include <iostream>

#include <sstream>

using namespace std;

template <class T>

class SLinkedList {

public:

class Node; // Forward declaration

protected:

Node\* head;

Node\* tail;

int count;

public:

SLinkedList()

{

this->head = nullptr;

this->tail = nullptr;

this->count = 0;

}

~SLinkedList(){};

void add(T e)

{

Node \*pNew = new Node(e);

if (this->count == 0)

{

this->head = this->tail = pNew;

}

else

{

this->tail->next = pNew;

this->tail = pNew;

}

this->count++;

}

int size()

{

return this->count;

}

void printList()

{

stringstream ss;

ss << "[";

Node \*ptr = head;

while (ptr != tail)

{

ss << ptr->data << ",";

ptr = ptr->next;

}

if (count > 0)

ss << ptr->data << "]";

else

ss << "]";

cout << ss.str() << endl;

}

public:

class Node {

private:

T data;

Node\* next;

friend class SLinkedList<T>;

public:

Node() {

next = 0;

}

Node(T data) {

this->data = data;

this->next = nullptr;

}

};

void bubbleSort();

};

**For example:**

| **Test** | **Result** |
| --- | --- |
| int arr[] = {9, 2, 8, 4, 1};  SLinkedList<int> list;  for(int i = 0; i <int(sizeof(arr))/4;i++)  list.add(arr[i]);  list.bubbleSort(); | [2,8,4,1,9]  [2,4,1,8,9]  [2,1,4,8,9]  [1,2,4,8,9] |

|  |
| --- |
| template <class T>  void SLinkedList<T>::bubbleSort()  {  for(int i = 0;i<this->count - 1; i++)  {  Node\*c=this->head;  for(int j=i+1; j <this->count;j++)  {  if((c->data)>(c->next->data))  {  int temp = c->data;  c->data = c->next->data;  c->next->data = temp;  }  c=c->next;  }  this->printList();  }  } |

|  | **Test** | **Expected** | **Got** |
| --- | --- | --- | --- |
|  | int arr[] = {9, 2, 8, 4, 1};  SLinkedList<int> list;  for(int i = 0; i <int(sizeof(arr))/4;i++)  list.add(arr[i]);  list.bubbleSort(); | [2,8,4,1,9]  [2,4,1,8,9]  [2,1,4,8,9]  [1,2,4,8,9] | [2,8,4,1,9]  [2,4,1,8,9]  [2,1,4,8,9]  [1,2,4,8,9] |  |

Passed all tests!

7. Implement static method selectionSortin class **Sorting**to sort an array in ascending order.  After each selection, we will print out a list to check (using printArray).

#include <iostream>

using namespace std;

template <class T>

class Sorting

{

public:

/\* Function to print an array \*/

static void printArray(T \*start, T \*end)

{

int size = end - start;

for (int i = 0; i < size - 1; i++)

cout << start[i] << ", ";

cout << start[size - 1];

cout << endl;

}

static void selectionSort(T \*start, T \*end);

};

**For example:**

| **Test** | **Result** |
| --- | --- |
| int arr[] = {9, 2, 8, 1, 0, -2};  Sorting<int>::selectionSort(&arr[0], &arr[6]); | -2, 2, 8, 1, 0, 9  -2, 0, 8, 1, 2, 9  -2, 0, 1, 8, 2, 9  -2, 0, 1, 2, 8, 9  -2, 0, 1, 2, 8, 9 |

|  |
| --- |
| template <class T>  void Sorting<T>::selectionSort(T \*start, T \*end)  {  int size = end - start;  for(int i = 0; i<size-1; i++)  {  int min=999;  int minPlace=i;  for(int j = i+1;j<size;j++)  {  if(min>start[j]) {min=start[j];minPlace=j;}  }  if(min<start[i]) {int temp = start[minPlace];  start[minPlace] = start[i];  start[i] = temp;  }  printArray(start,end);  }  } |

|  | **Test** | **Expected** | **Got** |
| --- | --- | --- | --- |
|  | int arr[] = {9, 2, 8, 1, 0, -2};  Sorting<int>::selectionSort(&arr[0], &arr[6]); | -2, 2, 8, 1, 0, 9  -2, 0, 8, 1, 2, 9  -2, 0, 1, 8, 2, 9  -2, 0, 1, 2, 8, 9  -2, 0, 1, 2, 8, 9 | -2, 2, 8, 1, 0, 9  -2, 0, 8, 1, 2, 9  -2, 0, 1, 8, 2, 9  -2, 0, 1, 2, 8, 9  -2, 0, 1, 2, 8, 9 |  |

Passed all tests!