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.

#include <iostream>

#include <sstream>

#include <string>

#include <assert.h>

using namespace std;

template <class T>

class SLinkedList {

public:

class Node; // Forward declaration

protected:

Node\* head;

Node\* tail;

int count;

public:

SLinkedList(): head(NULL), tail(NULL), count(0) {};

~SLinkedList() { this->clear(); };

void add(const T& e);

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

int size();

bool empty();

void clear();

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 &removeItem);

string toString() {

stringstream ss;

ss << "[";

Node\* ptr = head;

while(ptr != tail){

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

ptr = ptr->next;

}

if(count > 0)

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

else

ss << "]";

return ss.str();

}

public:

class Node {

private:

T data;

Node\* next;

friend class SLinkedList<T>;

public:

Node() {

next = 0;

}

Node(Node\* next) {

this->next = next;

}

Node(T data, Node\* next = NULL) {

this->data = data;

this->next = next;

}

};

};

template<class T>

void SLinkedList<T>::add(const 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++;

}

template<class T>

void SLinkedList<T>::add(int index, const T& e) {

if((index < 0) || (index > count))

throw std::out\_of\_range("Index is out of range");

if (index == this->count)

return this->add(e);

Node\* pNew = new Node(e);

if (index == 0) {

pNew->next = this->head;

this->head = pNew;

}

else {

Node\* pre = head;

int i = 0;

while (pre != NULL) {

if (i == index - 1) break;

pre = pre->next;

i++;

}

pNew->next = pre->next;

pre->next = pNew;

}

this->count++;

}

template<class T>

int SLinkedList<T>::size() {

return this->count;

}

template<class T>

T SLinkedList<T>::get(int index) {

/\* Give the data of the element at given index in the list. \*/

if ((index < 0) || (index > count - 1))

throw std::out\_of\_range("The index is out of range!");

Node \*temp = this->head;

int cursor = 0;

while (temp != NULL)

{

if (cursor == index)

break;

temp = temp->next;

cursor++;

}

return temp->data;

}

template <class T>

void SLinkedList<T>::set(int index, const T& e) {

/\* Assign new value for element at given index in the list \*/

if ((index < 0) || (index > count - 1))

throw std::out\_of\_range("The index is out of range!");

Node \*temp = this->head;

int cursor = 0;

while (temp != NULL)

{

if (cursor == index)

{

temp->data = e;

return;

}

temp = temp->next;

cursor++;

}

}

template<class T>

bool SLinkedList<T>::empty() {

/\* Check if the list is empty or not. \*/

return count == 0;

}

template<class T>

int SLinkedList<T>::indexOf(const T& item) {

/\* Return the first index wheter item appears in list, otherwise return -1 \*/

Node \*temp = this->head;

int cursor = 0;

while (temp != NULL)

{

if (temp->data == item)

return cursor;

temp = temp->next;

cursor++;

}

return -1;

}

template<class T>

bool SLinkedList<T>::contains(const T& item) {

/\* Check if item appears in the list \*/

return indexOf(item) != -1;

}

/// BEGIN <STUDENT ANSWER>

template <class T>

T SLinkedList<T>::removeAt(int index)

{

/\* Remove element at index and return removed value \*/

//TODO

}

template <class T>

bool SLinkedList<T>::removeItem(const T& item)

{

/\* Remove the first apperance of item in list and return true, otherwise return false \*/

//TODO

}

template<class T>

void SLinkedList<T>::clear(){

/\* Remove all elements in list \*/

//TODO

}

**For example:**

| **Test** | **Result** |
| --- | --- |
| SLinkedList<int> list;  for (int i = 0; i < 10; ++i) {  list.add(i);  }  assert(list.get(0) == list.removeAt(0));  cout << list.toString(); | [1,2,3,4,5,6,7,8,9] |
| SLinkedList<int> list;  for (int i = 0; i < 10; ++i) {  list.add(i);  }  assert(list.get(9) == list.removeAt(9));  cout << list.toString(); | [0,1,2,3,4,5,6,7,8] |
| SLinkedList<int> list;  for (int i = 0; i < 10; ++i) {  list.add(i);  }  assert(list.removeItem(9));  cout << list.toString(); | [0,1,2,3,4,5,6,7,8] |