BÁCH KHOA E-LEARNING

Đã bắt đầu vào lúc	Saturday, 17 April 2021, 9:57 AM
Tình trạng	Đã hoàn thành
Hoàn thành vào lúc	Saturday, 24 April 2021, 3:27 AM
Thời gian thực hiện	6 ngày 17 giờ
Điểm	**2,90** của 3,00 (**97**%)

Câu hỏi **1**

Chính xác

Điểm 1,00 của 1,00

Implement methods **add, size** 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 SLinkedList {
public:
    class Node; // Forward declaration
protected:
    Node* head;
    Node* tail;
    int count;
public:
    SLinkedList();
    ~SLinkedList();
    void    add(T e);
    void    add(int index, T e);
    int     size();
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) {
            this->data = data;
            this->next = next;
        }
    };
};
```

For example:

Test	Result

`SLinkedList<int> list;`
`int size = 10;`
 	
`for(int index = 0; index < size; index++){`
`  list.add(index);`
`}`
 	
`cout << list.toString();`
	`[0,1,2,3,4,5,6,7,8,9]`

Answer: (penalty regime: 0 %)

Reset answer

```
1  template <class T>
2  void SLinkedList<T>::add(const T& e) {
3      // allocating space
4      T data=e;
5      Node* newNode = new Node();
6
7      // inserting the required data
```

```
    newNode->data = data;
    newNode->next = NULL;

    // If the Linked List is empty, then make the new node as head
if (head == NULL)
    {
        ////////////////////////////////
        head = newNode;
        tail = newNode;
    }
    else {
    tail->next=newNode;
    tail=newNode;}
count++;
}
////////////////////////////////////////////////////////////////////////////////////
template <class T>
void SLinkedList<T>::add(int index, const T& e) {

    int pos = index; //?//
    Node** current = &head;
    T data = e;

    if (pos > count ) pos = count;
    if (pos < 0) pos = 0;

    // allocating space
    Node* newNode = new Node();

    // inserting the required data
    newNode->data = data;
    newNode->next = NULL;

    // allocating space
    Node* temp = new Node();

    // inserting the required data
    temp->data = data;
    temp->next = NULL;


    // If the Linked List is empty, then make the new node as head
if (head == NULL)
    {
        ////////////////////////////////
        head = newNode;
        tail = newNode;
    }

if (pos == 0)
{
    newNode->next = head; //link to old head
    head = newNode; //then become new head
}
else if (pos == count) {
add(data);
count--;
}

else {

            // Keep looping until the pos is zero
            for(int i=0;i<pos;i++) {
             current = &(*current)->next;
            }
            temp->next = *current;
            *current = temp;
        }

count++;
}

template<class T>
int SLinkedList<T>::size() {
    if (head == NULL)  count =0;

    return count;
}

```

	Test	Expected	Got	

✔	SLinkedList<int> list;
	int size = 10;
	
	for(int index = 0; index < size; index++){
	  list.add(index);
	}
	
	cout << list.toString();
		[0,1,2,3,4,5,6,7,8,9]	[0,1,2,3,4,5,6,7,8,9]	✔

✔	SLinkedList<int> list;
	int size = 10;
	
	for(int index = 0; index < size; index++){
	  list.add(0, index);
	}
	
	cout << list.toString();
		[9,8,7,6,5,4,3,2,1,0]	[9,8,7,6,5,4,3,2,1,0]	✔

✔	SLinkedList<int> list;
	int size = 10;
	
	for (int index = 0; index < size; index++) {
	  list.add(list.size(), index);
	}
	
	cout << list.toString();
		[0,1,2,3,4,5,6,7,8,9]	[0,1,2,3,4,5,6,7,8,9]	✔

✔	SLinkedList<int> list;
	int values[] = {10, 15, 2, 6, 4, 7, 40, 8};
	int index[] = {0, 0, 1, 3, 2, 3, 5, 0};
	for (int idx = 0; idx < 8; idx++){
	  list.add(index[idx], values[idx]);
	}
	
	cout << list.toString();
		[8,15,2,4,7,10,40,6]	[8,15,2,4,7,10,40,6]	✔

✔	SLinkedList<int> list;
	int values[] = {10, 15, 2, 6, 4, 7, 40, 8};
	int index[] = {0, 0, 1, 3, 2, 3, 5, 0};
	
	for (int idx = 0; idx < 8; idx++){
	  list.add(index[idx], values[idx]);
	}
	
	cout << list.size();
		8	8	✔

✔	SLinkedList<int> list;
	list.add(0);
	
	for (int i= 0; i< 10; i++) {
	  list.add(list.size() - 1, list.size());
	}
	
	cout << list.toString();
		[1,2,3,4,5,6,7,8,9,10,0]	[1,2,3,4,5,6,7,8,9,10,0]	✔

✔	SLinkedList<int> list;
	
	int values[] = { 13, 23, 7, 9, 8, 7, 50, -1 };
	int index[] = { 0, 1, 1, 3, 2, 3, 5, 1 };
	
	for (int idx = 0; idx < 8; idx++) {
	  list.add(index[idx], values[idx]);
	}
	
	cout << list.toString();
		[13,-1,7,8,7,23,50,9]	[13,-1,7,8,7,23,50,9]	✔

	Test	Expected	Got	

✔	`SLinkedList<int> list;`
	
	`int values[] = { 13, 23, 7, 9, 8, 7, 50, -1 };`
	`int index[] = { 0, 1, 1, 3, 2, 3, 5, 1 };`
	
	`for (int idx = 0; idx < 8; idx++) {`
	`  list.add(index[idx], values[idx]);`
	`}`
	
	`cout << list.size();`
		8	8	✔

✔	`SLinkedList<int> list;`
	
	`for (int i = 0; i < 10; i++) {`
	`  if (i % 2 == 0) list.add(0, i);`
	`  else list.add(list.size(), i);`
	`}`
	
	`cout << list.size();`
		10	10	✔

✔	`SLinkedList<int> list;`
	
	`for (int i = 0; i < 10; i++) {`
	`  if (i % 2 == 0) list.add(0, i);`
	`  else list.add(list.size(), i);`
	`}`
	
	`cout << list.toString();`
		[8,6,4,2,0,1,3,5,7,9]	[8,6,4,2,0,1,3,5,7,9]	✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi **2**

Chính xác

Điểm 1,00 của 1,00

Implement methods **get, set, empty, indexOf, contains** 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 SLinkedList {
public:
    class Node; // Forward declaration
protected:
    Node* head;
    Node* tail;
    int count;
public:
    SLinkedList(): head(NULL), tail(NULL), count(0);
    ~SLinkedList() { };
    void    add(T e);
    void    add(int index, T e);
    int     size();
    bool    empty();
    T       get(int index);
    void    set(int index, T e);
    int     indexOf(T item);
    bool    contains(T item);
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;
        }
    };
};
```

For example:

Test	Result

Test	Result

SLinkedList<int> list;
int values[] = {10, 15, 2, 6, 4, 7, 40, 8};
int index[] = {0, 0, 1, 3, 2, 3, 5, 0};
int expvalues[]= {8, 15, 2, 4, 7, 10, 40, 6};
 	
for (int idx = 0; idx < 8; idx++){
list.add(index[idx], values[idx]);
}
 	
assert(list.size() == 8);
 	
for (int idx = 0; idx < 8; idx++){
assert(list.get(idx) == expvalues[idx]);
}
 	
cout << list.toString();
	[8,15,2,4,7,10,40,6]

Answer: (penalty regime: 10, 20, ... %)

Reset answer

```
template<class T>
T SLinkedList<T>::get(int index) {
    /* Give the data of the element at given index in the list. */

    Node* tmp=new Node;
    tmp=head;
    if(index<0||index>(count-1)){
    throw std::out_of_range("F");
    }
    for(int i = 0; i< index; i++)    tmp=tmp->next;
    return tmp->data;

}




template <class T>
void SLinkedList<T>::set(int index, const T& e) {
    /* Assign new value for element at given index in the list */
        Node* tmp=new Node;
    tmp=head;
        if(index<0||index>(count-1)){
    throw std::out_of_range("F");
    }
    for(int i=0;i<index;i++){
    tmp=tmp->next;}
    tmp->data=e;
}




template<class T>
bool SLinkedList<T>::empty() {
    /* Check if the list is empty or not. */
     if(head==NULL)return true;
     else return false;
}



template<class T>
int SLinkedList<T>::indexOf(const T& item) {
    /* Return the first index wheter item appears in list, otherwise return -1 */
   Node* temp = head;
    int coun = 0;
    while(temp!= NULL)
    {
       if (temp->data == item)  return coun;
       ++coun;
       temp = temp->next;
    }
    //return -1 if no match
```

```
    //return -1 if no match
    return -1;
}



template<class T>
bool SLinkedList<T>::contains(const T& item) {
    Node* temp = head;
    while(temp!= NULL)
    {
       if (temp->data == item)  return 1;
       temp = temp->next;
    }
    return 0;
}
```

	Test	Expected	Got	

✔	`SLinkedList<int> list;`
	`int values[]  = {10, 15, 2,  6,  4,  7,  40, 8};`
	`int index[]  = {0,  0,  1,  3,  2,  3,  5, 0};`
	`int expvalues[]= {8,  15, 2,  4,  7, 10,  40, 6};`
	
	`for (int idx = 0; idx < 8; idx++){`
	`  list.add(index[idx], values[idx]);`
	`}`
	
	`assert( list.size() == 8 );`
	
	`for (int idx = 0; idx < 8; idx++){`
	`  assert( list.get(idx) == expvalues[idx] );`
	`}`
	
	`cout << list.toString();`
		`[8,15,2,4,7,10,40,6]`	`[8,15,2,4,7,10,40,6]`	✔

✔	`SLinkedList<int> list;`
	
	`assert( list.empty() == true );`
	`cout << list.toString();`
		`[]`	`[]`	✔

✔	`SLinkedList<int> list;`
	`for (int i = 0; i < 10; ++i) {`
	`  list.add(i);`
	`  }`
	
	`assert( list.empty() == false );`
	`cout << list.toString();`
		`[0,1,2,3,4,5,6,7,8,9]`	`[0,1,2,3,4,5,6,7,8,9]`	✔

✔	`SLinkedList<int> list;`
	
	`for (int i = 0; i < 10; ++i) {`
	`  list.add(i);`
	`}`
	`for (int i = 0; i < 10; ++i) {`
	`  assert(list.indexOf(i) == i);`
	`}`
	
	`cout << list.toString();`
		`[0,1,2,3,4,5,6,7,8,9]`	`[0,1,2,3,4,5,6,7,8,9]`	✔

	Test	Expected	Got	

✔	`SLinkedList<int> list;`
	
	`for (int i = 0; i < 10; ++i) {`
	`  list.add(i);`
	`}`
	`for (int i = 0; i < 10; ++i) {`
	`  assert( list.contains(i) == true );`
	`}`
	
	`cout << list.toString();`
		`[0,1,2,3,4,5,6,7,8,9]`	`[0,1,2,3,4,5,6,7,8,9]`	✔

✔	`SLinkedList<int> list;`
	
	`for (int i = 0; i < 10; ++i) {`
	`  list.add(i);`
	`}`
	
	`for (int i = 10; i < 20; ++i) {`
	`  assert(list.indexOf(i) == -1);`
	`}`
	
	`cout << list.toString();`
		`[0,1,2,3,4,5,6,7,8,9]`	`[0,1,2,3,4,5,6,7,8,9]`	✔

✔	`SLinkedList<int> list;`
	
	`for (int i = 0; i < 10; ++i) {`
	`  list.add(i);`
	`}`
	`for (int i = 10; i < 20; ++i) {`
	`  assert( list.contains(i) == false );`
	`}`
	
	`cout << list.toString();`
		`[0,1,2,3,4,5,6,7,8,9]`	`[0,1,2,3,4,5,6,7,8,9]`	✔

✔	`SLinkedList<int> list;`
	
	`for (int i = 0; i < 10; ++i) {`
	`  list.add(i);`
	`}`
	`for (int i = 0; i < 10; ++i) {`
	`  list.set(i, i + 10);`
	`}`
	`for (int i = 0; i < 10; ++i) {`
	`  assert(list.get(i) == i + 10);`
	`}`
	
	`cout << list.toString();`
		`[10,11,12,13,14,15,16,17,18,19]`	`[10,11,12,13,14,15,16,17,18,19]`	✔

✔	`SLinkedList<int> list;`
	
	`for (int i = 0; i < 10; ++i) {`
	`  list.add(i);`
	`}`
	
	`try {`
	`  list.get(100);`
	`}`
	`catch(std::out_of_range e){`
	`  assert( 1 == 1 ); //pass`
	`  e.what();`
	`}`
				✔

	Test	Expected	Got	

✔	`SLinkedList<int> list;`
	
	`for (int i = 0; i < 10; ++i) {`
	`list.add(i);`
	`}`
	
	`try {`
	`list.set(100, 100);`
	`}`
	`catch (std::out_of_range e) {`
	`assert(1 == 1); //pass`
	`e.what();`
	`}`
				✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi **3**

Đúng một phần

Điểm 0,90 của 1,00

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 SLinkedList {
public:
    class Node; // Forward declaration
protected:
    Node* head;
    Node* tail;
    int count;
public:
    SLinkedList();
    ~SLinkedList();
    void    add(T e);
    void    add(int index, T e);
    int     size();
    bool    empty();
    int     size();
    void    clear();
    T       get(int index);
    void    set(int index, T e);
    int     indexOf(T item);
    bool    contains(T item);
    T       removeAt(int index);
    bool    removeItem(T item);
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;
        }
    };
};
```

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]`

Answer: (penalty regime: 10, 20, ... %)

Reset answer

```
////////////////////////////////////////////////////////
template <class T>
T SLinkedList<T>::removeAt(int index)
{
    /* Remove element at index and return removed value */
    T tdata;
    // If linked list is empty
    if (head == NULL) throw"nothing to delete";

    // Store head node
    Node* temp = head;

    // If head needs to be removed
    if (index == 0)
    {

        // Change head
        head = temp->next;

        // Free old head
        T tdata = temp->data;
        delete(temp);
        --count;
        return tdata;
    }

    // Find previous node of the node to be deleted
    for (int i = 0; temp != NULL && i < index - 1; i++)
        temp = temp->next;

    // If index is more than number of nodes
   // if (temp == NULL || temp->next == NULL) throw"out of range";

    // Node temp->next is the node to be deleted
    // Store pointer to the next of node to be deleted
    Node* next = temp->next->next;

    // Unlink the node from linked list
    tdata = temp->next->data;
    delete(temp->next); // Free memory

    // Unlink the deleted node from list
    temp->next = next;
    --count;
    return tdata;
}

template <class T>
bool SLinkedList<T>::removeItem(const T& item)
{
    /* Remove the first apperance of item in list and return true, otherwise return false */

    int i;
    /* Remove the first apperance of item in list and return true, otherwise return false */
    i = SLinkedList::indexOf(item);
    if (i == -1) return false;
    else
    {
        removeAt(i);
        return true;
    }

}



template<class T>
void SLinkedList<T>::clear() {
    /* Remove all elements in list */
    for (int i = 0; i < count; i++) {
        removeAt(i);
    }
    count = 0;
}
```

	Test	Expected	Got	

✔	`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]`	`[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]`	`***Error***`
			`Segmentation fault (core dumped)`
				✖

Testing was aborted due to error.

Show differences

Đúng một phần

Điểm cho bài nộp này: 0,10/1,00. Tính toán cho lần làm bài trước đó, điểm **0,90/1,00**.

◄ Data Structures and Algorithms (Lab) - CC02 - 1PM to 3:50PM - 25/05/2021

Chuyển tới...

Test ►