|  |
| --- |
| #include <iostream> |
|  | using namespace std; |
|  |  |
|  | //Defining the basic structure of a node. |
|  | struct node{ |
|  | int data; |
|  | node \*next, \*previous; |
|  | }; |
|  |  |
|  | //Creating a doubly linked list class |
|  | class doublyLinkedList{ |
|  | private: |
|  | node \*head, \*tail; |
|  | public: |
|  | //Constructor that sets both head and tail to NULL. |
|  | doublyLinkedList() { |
|  | head = NULL; |
|  | tail = NULL; |
|  | } |
|  | //Defining functions to operate on the linked list. |
|  | void Insert(int value); |
|  | void InsertAt(int pos, int value); |
|  | void Delete(); |
|  | void DeleteAt(int pos); |
|  | int countItems(); |
|  | void display(); |
|  | }; |
|  |  |
|  | //Function to count the number of items in the linked list |
|  | int doublyLinkedList::countItems() { |
|  | if (head == NULL) { |
|  | return 0; |
|  | } |
|  | else { |
|  | node \*temp; |
|  | temp = head; |
|  | int count = 0; |
|  | while (temp != NULL) { |
|  | temp = temp->next; |
|  | count++; |
|  | } |
|  | return count; |
|  | } |
|  | } |
|  |  |
|  | //Function to insert element at the end |
|  | void doublyLinkedList::Insert(int value) { |
|  | node \*temp = new node; |
|  | temp->data = value; |
|  | temp->next = NULL; |
|  | temp->previous = NULL; |
|  | //If the linked list is empty, the new node becomes both the head and the tail |
|  | if (head == NULL) { |
|  | head = temp; |
|  | tail = temp; |
|  | } |
|  | else { |
|  | //Adding the new node to the end of the linked list |
|  | tail->next = temp; |
|  | temp->previous = tail; |
|  | //Changing the tail to the new last element |
|  | tail = temp; |
|  | } |
|  | } |
|  |  |
|  | //Function to delete element from the end |
|  | void doublyLinkedList::Delete() { |
|  | //When the linked list is empty |
|  | if (head == NULL) { |
|  | cout << "The linked list is empty and has no elements to delete." << endl; |
|  | } |
|  | //When there's only one element in the linked list |
|  | else if (countItems() == 1) { |
|  | head = NULL; |
|  | tail = NULL; |
|  | } |
|  | //For all other cases |
|  | else { |
|  | node \*temp = tail; |
|  | //Now we change tail to the second last element of the linked list |
|  | tail = tail->previous; |
|  | //Remove the links between tail and temp |
|  | tail->next = NULL; |
|  | temp->previous = NULL; |
|  | //Delete the pointer temp |
|  | delete temp; |
|  | } |
|  | } |
|  |  |
|  | //Function to insert element at a specific position |
|  | void doublyLinkedList::InsertAt(int pos, int value) { |
|  | node \*preceeding, \*succeeding; |
|  | node \*temp = new node; |
|  | temp->data = value; |
|  | temp->next = NULL; |
|  | temp->previous = NULL; |
|  | succeeding = head; |
|  | preceeding = NULL; |
|  | //If the linked list is empty, we add value as the first element. |
|  | if (head == NULL) { |
|  | Insert(value); |
|  | cout << "The linked list is empty. " << value << " has been added as the first element.\n"; |
|  | } |
|  | //When the element needs to be added at the first position... |
|  | else if (pos == 1) { |
|  | temp->next = head; |
|  | head->previous = temp; |
|  | head = temp; |
|  | } |
|  | //For all other cases... |
|  | else { |
|  | //A loop that brings the pointers preceeding and succeeding to the required positions |
|  | for (int i = 1; i < pos; i++) { |
|  | preceeding = succeeding; |
|  | succeeding = succeeding->next; |
|  | } |
|  | if (preceeding == NULL) { |
|  | cout << "The linked list doesn't have that many elements. " << value << " has been added at the end of the list.\n"; |
|  | Insert(value); |
|  | } |
|  | else { |
|  | //Create the links between preceeding and temp |
|  | preceeding->next = temp; |
|  | temp->previous = preceeding; |
|  | //and now between temp and succeeding, thereby introducing it between the two |
|  | temp->next = succeeding; |
|  | if (succeeding != NULL) { |
|  | succeeding->previous = temp; |
|  | } |
|  | else { |
|  | tail = temp; |
|  | } |
|  | } |
|  | } |
|  | } |
|  |  |
|  | //Function to delete the element at a specific position |
|  | void doublyLinkedList::DeleteAt(int pos) { |
|  | //Create three pointers |
|  | node \*current; |
|  | current = head; |
|  | //When the linked list is empty |
|  | if (head == NULL) { |
|  | cout << "The linked list is empty and has no elements to delete." << endl; |
|  | } |
|  | //When the linked list is smaller than the position mentioned |
|  | else if (countItems() < pos) { |
|  | cout << "The linked list has no element in the position " << pos << "." << endl; |
|  | } |
|  | //When the first element needs to be deleted |
|  | else if (pos == 1) { |
|  | head = head->next; |
|  | current->next = NULL; |
|  | head->previous = NULL; |
|  | delete current; |
|  | } |
|  | //For all other cases |
|  | else { |
|  | //We bring 'current' to the required position |
|  | for (int i = 1; i < pos; i++) { |
|  | current = current->next; |
|  | } |
|  | //Link the node preceeding current to the one succeeding it |
|  | current->previous->next = current->next; |
|  | if (current->next != NULL) { |
|  | current->next->previous = current->previous; |
|  | } |
|  | //Remove the links that the node has to the rest of the linked list |
|  | current->next = NULL; |
|  | current->previous = NULL; |
|  | //Delete the pointer |
|  | delete current; |
|  | } |
|  | } |
|  |  |
|  | //Function to display the elements of the linked list |
|  | void doublyLinkedList::display() { |
|  | node \*temp; |
|  | temp = head; |
|  | //When the linked list has no elements |
|  | if (head == NULL) { |
|  | cout << "The linked list has no elements.\n"; |
|  | } |
|  | //When the linked list is not empty |
|  | else { |
|  | while (temp != NULL) { |
|  | //Display the data stored in the current element... |
|  | cout << temp->data << " --> "; |
|  | //and then move to the next one. |
|  | temp = temp->next; |
|  | } |
|  | cout << "NULL" << endl; |
|  | } |
|  | } |
|  |  |
|  | int main() { |
|  | //And now you're free to do what you want...:) |
|  | doublyLinkedList one; |
|  | one.Insert(5); |
|  | one.InsertAt(2, 4); |
|  | one.DeleteAt(2); |
|  | one.display(); |
|  | cout << endl; |
|  | return 0; |
|  | } |