SINGLY LINKED LIST CODE IMPLEMENTATION

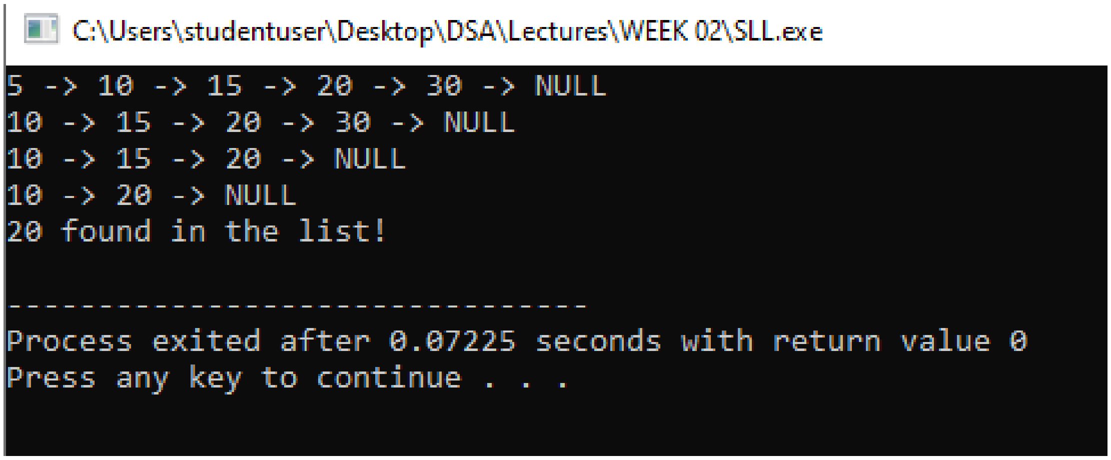
```
#include <iostream>
using namespace std;


// Node class
class Node {
public:
   int data;    // Data field
   Node* next;  // Pointer to the next node


   // Constructor
   Node(int value) {
      data = value;
      next = NULL;  // Initialize the next pointer to nullptr
   }
};


// Singly Linked List class
class SinglyLinkedList {
private:
   Node* head;  // Head pointer


public:
   // Constructor
   SinglyLinkedList() {
    head = NULL;  // Initialize the head to nullptr (empty list)
}

// Function to insert a node at the beginning of the list
void insertAtBeginning(int value) {
    Node* newNode = new Node(value);  // Create a new node
    newNode->next = head;  // Point the new node to the current head
    head = newNode;  // Update the head to the new node
}

// Function to insert a node at the end of the list
void insertAtEnd(int value) {
    Node* newNode = new Node(value);  // Create a new node

    if (head == NULL) {  // If the list is empty, new node becomes the head
        head = newNode;
    } else {
        Node* temp = head;  // Start from the head
        while (temp->next != NULL) {  // Traverse to the last node
            temp = temp->next;
        }
        temp->next = newNode;  // Set the last node's next to the new node
    }
}

// Function to insert a node at any position in the list
void insertAtPosition(int value, int position) {
        Node* newNode = new Node(value);  // Create a new node

        if (position == 1) {  // If the position is at the beginning
            newNode->next = head;
            head = newNode;
            return;
        }

        Node* temp = head;
        for (int i = 1; i < position - 1 && temp != NULL; i++) {  // Traverse to the node before the desired position
            temp = temp->next;
        }

        if (temp == NULL) {  // If the position is beyond the current length
            cout << "Position out of bounds!" << endl;
            delete newNode;
        } else {
            newNode->next = temp->next;  // Insert the new node
            temp->next = newNode;
        }
    }

    // Function to delete a node from the beginning of the list
    void deleteFromBeginning() {
        if (head == NULL) {  // If the list is empty
            cout << "List is empty!" << endl;
        return;
    }
    Node* temp = head;  // Store the current head
    head = head->next;  // Move the head to the next node
    delete temp;  // Delete the old head
}

// Function to delete a node from the end of the list
void deleteFromEnd() {
    if (head == NULL) {  // If the list is empty
        cout << "List is empty!" << endl;
        return;
    }

    if (head->next == NULL) {  // If there is only one node
        delete head;
        head = NULL;
    } else {
        Node* temp = head;  // Start from the head
        while (temp->next->next != NULL) {  // Traverse to the second last node
            temp = temp->next;
        }
        delete temp->next;  // Delete the last node
        temp->next = NULL;  // Set the new last node's next to nullptr
    }
}
```

```
// Function to delete a node from any position in the list
void deleteFromPosition(int position) {
    if (head == NULL) {  // If the list is empty
        cout << "List is empty!" << endl;
        return;
    }

    if (position == 1) {  // If the position is at the beginning
        deleteFromBeginning();
        return;
    }

    Node* temp = head;
    for (int i = 1; i < position - 1 && temp->next != NULL; i++) {  // Traverse to the node before the desired position
        temp = temp->next;
    }

    if (temp->next == NULL) {  // If the position is beyond the current length
        cout << "Position out of bounds!" << endl;
    } else {
        Node* nodeToDelete = temp->next;
        temp->next = nodeToDelete->next;  // Remove the node from the list
        delete nodeToDelete;
    }
}
```

```
  // Function to traverse and display the list
  void traverse() {
    if (head == NULL) {  // If the list is empty
      cout << "List is empty!" << endl;
      return;
    }

    Node* temp = head;  // Start from the head
    while (temp != NULL) {  // Traverse the list
      cout << temp->data << " -> ";  // Print the data of each node
      temp = temp->next;  // Move to the next node
    }
    cout << "NULL" << endl;  // End of the list
  }

  // Function to search for a value in the list
  bool search(int value) {
    Node* temp = head;  // Start from the head
    while (temp != NULL) {  // Traverse the list
      if (temp->data == value) {  // If the value is found
        return true;
      }
      temp = temp->next;  // Move to the next node
    }
    return false;  // Value not found
  }
};
```

```
int main() {
    SinglyLinkedList list;

    // Insertion operations
    list.insertAtEnd(10);
    list.insertAtEnd(20);
    list.insertAtEnd(30);
    list.insertAtBeginning(5);
    list.insertAtPosition(15, 3);  // Inserting 15 at position 3

    // Traversal
    list.traverse();  // Output: 5 -> 10 -> 15 -> 20 -> 30 -> NULL

    // Deletion operations
    list.deleteFromBeginning();
    list.traverse();  // Output: 10 -> 15 -> 20 -> 30 -> NULL

    list.deleteFromEnd();
    list.traverse();  // Output: 10 -> 15 -> 20 -> NULL

    list.deleteFromPosition(2);  // Deleting node at position 2
    list.traverse();  // Output: 10 -> 20 -> NULL

    // Search operation
    if (list.search(20)) {
        cout << "20 found in the list!" << endl;
    } else {
        cout << "20 not found in the list!" << endl;
    }

    return 0;
}
```

OUTPUT



C:\Users\studentuser\Desktop\DSA\Lectures\WEEK 02\SLL.exe

```
5 -> 10 -> 15 -> 20 -> 30 -> NULL
10 -> 15 -> 20 -> 30 -> NULL
10 -> 15 -> 20 -> NULL
10 -> 20 -> NULL
20 found in the list!

--------------------------------
Process exited after 0.07225 seconds with return value 0
Press any key to continue . . .
```