# Lab Task 4: Singly Linked List (Insert at Specific Position)

## Code Explanation:

This program implements a Singly Linked List with three types of insertions:  
- \*\*Insert at Start:\*\* A new node is added at the beginning of the list.  
- \*\*Insert at End:\*\* A new node is added at the end of the list.  
- \*\*Insert at Specific Position:\*\* A new node is inserted at a given position.  
- The `display()` function prints the linked list after each insertion.  
- The `main()` function inserts values `{3, 1, 5, 7}` sequentially and then inserts `{4, 6}` at positions `2` and `5`.

## Code:

#include <iostream>  
using namespace std;  
  
class Node {  
public:  
 int data;  
 Node\* next;  
 Node(int val) : data(val), next(nullptr) {}  
};  
  
class SinglyLinkedList {  
public:  
 Node\* head;  
 SinglyLinkedList() : head(nullptr) {}  
  
 void insertAtStart(int val) {  
 Node\* newNode = new Node(val);  
 newNode->next = head;  
 head = newNode;  
 display();  
 }  
  
 void insertAtEnd(int val) {  
 Node\* newNode = new Node(val);  
 if (!head) {  
 head = newNode;  
 } else {  
 Node\* temp = head;  
 while (temp->next) {  
 temp = temp->next;  
 }  
 temp->next = newNode;  
 }  
 display();  
 }  
  
 void insertAtPosition(int val, int pos) {  
 if (pos < 1) {  
 cout << "Invalid position!" << endl;  
 return;  
 }  
 Node\* newNode = new Node(val);  
 if (pos == 1 || !head) {  
 insertAtStart(val);  
 return;  
 }  
 Node\* temp = head;  
 for (int i = 1; temp && i < pos - 1; i++) {  
 temp = temp->next;  
 }  
 if (!temp) {  
 cout << "Position out of bounds!" << endl;  
 return;  
 }  
 newNode->next = temp->next;  
 temp->next = newNode;  
 display();  
 }  
  
 void display() {  
 Node\* temp = head;  
 while (temp) {  
 cout << temp->data << " -> ";  
 temp = temp->next;  
 }  
 cout << "NULL" << endl;  
 }  
};  
  
int main() {  
 SinglyLinkedList list;  
 list.insertAtStart(3);  
 list.insertAtStart(1);  
 list.insertAtEnd(5);  
 list.insertAtEnd(7);  
 list.insertAtPosition(4, 2);  
 list.insertAtPosition(6, 5);  
 return 0;  
}

## Output:

```  
3 -> NULL  
1 -> 3 -> NULL  
1 -> 3 -> 5 -> NULL  
1 -> 3 -> 5 -> 7 -> NULL  
1 -> 4 -> 3 -> 5 -> 7 -> NULL  
1 -> 4 -> 3 -> 5 -> 6 -> 7 -> NULL  
```

## Time Complexity:

- \*\*Insert at Start:\*\* O(1) (Constant time insertion at the beginning)  
- \*\*Insert at End:\*\* O(n) (Traversal required to reach the end of the list)  
- \*\*Insert at Position:\*\* O(n) (Traversal required to find the position)