DSA_Unit 2_Lecture 8 Coding
Test Summary

- No. of Sections: 1
- No. of Questions: 12
- Total Duration: 90 min

Section 1 - Coding
Section Summary

- No. of Questions: 12
- Duration: 90 min

Additional Instructions:

None
Q1.
Problem Statement

You need to write a program to maintain a student roster using a singly linked list.

Each student is represented by a node in the linked list, which contains the student's name. Implement an **insertNode()** function, which inserts a new student node with the given name at the **specified position** in the roster. The position is 1-based, meaning the first student is at position 1.

If the specified position is 1, the new student node should become the new head of the roster. If the specified position is greater than the current size of the roster, the new student node should be appended at the end.

After performing the insertion, you should print the contents of the roster.

Note: This is a sample question asked in an Amazon interview.

Input Format
The first line of input consists of an integer **n**, representing the number of students currently in the roster.

The next **n** lines of input consist of the names of the students on the roster, separated by a line.

The last line of input consists of a string name, representing the name of the new student to be inserted, and an integer, representing the position at which the new student should be inserted, separated by space.

Output Format
The first line of output should print the current linked list elements, separated by space.

The next line of output should print the updated linked list elements after the insertion at the given position, separated by space.

Refer to the sample output for formatting specifications.
Constraints
The student names consist of at most 100 characters.

The number of students on the roster should not exceed 100.

Sample Input Sample Output

```
4
John
Alice
Bob
Emma
Michael 3

Current Linked List:
John Alice Bob Emma
Updated Linked List:
John Alice Michael Bob Emma
```

Sample Input Sample Output

```
4
Emma
Daniel
Sophia
Oliver
Charlotte 1

Current Linked List:
Emma Daniel Sophia Oliver
Updated Linked List:
Charlotte Emma Daniel Sophia Oliver
```

Sample Input Sample Output

```
0
Emma 1

Current Linked List:

Updated Linked List:
Emma
```

Time Limit: - ms Memory Limit: - kb Code Size: - kb
Q2.
Problem Statement

You are tasked with implementing a program to manage a singly linked list. Each node in the linked list will store a string value.

Your goal is to write a function called **insertNode()** that inserts a new node at a specified position within the linked list. The position can be 1 (indicating insertion at the beginning), a positive integer (indicating insertion at the corresponding position), or a position greater than the size of the list (indicating appending at the end).

Once the insertion is completed, you need to print the contents of the updated linked list.

Note: This is a sample question asked in a TCS interview.
Input Format
The first line of input consists of an integer **n**, denoting the number of strings to be inserted initially.

The second line of input consists of **n** string values, separated by space for each node in the initial linked list.

The third line of input consists of a string denoting the data to be inserted into the linked list, and an integer **p**, denoting the position where the new node is to be inserted.

Output Format
The first line of output should print the current linked list elements, separated by space.

The next line of output should print the updated linked list elements, after the insertion, separated by space.

Refer to the sample output for formatting specifications.

Constraints
1 <= n <= 100

1 <= p <= n+1

The string values in the linked list will have at most 100 characters.

The position for insertion will be an integer between 1 and the size of the linked list + 1.

Sample Input Sample Output

```
6
A B C D E F
G 7
```

```
Current Linked List:
A B C D E F
Updated Linked List:
A B C D E F G
```

Sample Input Sample Output

```
3
Apple Orange Banana
Mango 1
```

```
Current Linked List:
Apple Orange Banana
Updated Linked List:
Mango Apple Orange Banana
```

Sample Input Sample Output

```
0

Watermelon 1
```

```
Current Linked List:

Updated Linked List:
Watermelon
```

Sample Input Sample Output

```
5
Dog Cat Bird Fish Monkey
Lion 6
```

```
Current Linked List:
Dog Cat Bird Fish Monkey
Updated Linked List:
Dog Cat Bird Fish Monkey Lion
```

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q3.

Problem Statement

Alice is learning about linked lists and wants to practice creating one by **inserting elements at the front**. She decides to write a program that allows her to do so.

The program will prompt Alice to enter the number of elements she wants to insert and then ask her to input the values of those elements. After receiving the inputs, the program will insert the elements at the front of the linked list and display the final linked list to Alice.

Alice is excited to see how her program will work and hopes it will help solidify her understanding of linked lists. She prepares herself to input the number of elements and their respective values, eagerly anticipating the final result.

Note: This is a sample question asked in a HCL interview.

Input Format

The first line of input consists of an integer **n,** representing the number of elements to be inserted.

The second line of input consists of **n** space-separated integers, representing the elements to be inserted in the linked list.

The third line of input consists of an integer representing the value to be inserted at the front of the linked list.

Output Format

The first line of output should display the initial linked list elements, separated by space.

The second line of output should display the linked list elements, after inserting the given element at the front, separated by space.

Refer to the sample output for formatting specifications.

Constraints

The number of elements to be inserted (n) is a positive integer.

The values of the elements can be positive or negative integers.
Sample Input Sample Output

```
5
6 5 4 3 2
1

Created Linked list: 2 3 4 5 6
Final List: 1 2 3 4 5 6
```

Sample Input Sample Output

```
4
-10 20 -30 40
5

Created Linked list: 40 -30 20 -10
Final List: 5 40 -30 20 -10
```

Time Limit: - ms Memory Limit: - kb Code Size: - kb
Q4.
Problem Statement

You are developing a text editor application that allows users to manage a document. The application uses a linked list data structure to represent the document content. Each node of the linked list contains a string representing a line of text.

The application supports two operations: **inserting new string values at the beginning** of the document and **appending a new string at the end** of the document.

If no input string list is created and if no value is appended to the list, then an empty list should be returned as output.

Note: This is a sample question asked in a Capgemini interview.

Input Format
The first line of input consists of an integer, **n** representing the number of lines in the document.

The next **n** lines of input consist of the lines of text that constitute the document.

The last line of input consists of a string, **s**, which needs to be appended at the end of the document.

Output Format
The first line of output should print the initial document content, which inserts the given **n** values at the beginning.

The second line of output should print the final document content, which appends the given value **s** at the end of the document.

Refer to the sample output for formatting specifications.

Constraints
n>0

The length of each line should be less than or equal to 100 characters.

Sample Input Sample Output

```
3
Apple
Banana
Orange
Grapes

Document: Orange Banana Apple
Updated Document: Orange Banana Apple Grapes
```

Sample Input Sample Output

```
2
Hello
World
Space

Document: World Hello
Updated Document: World Hello Space
```

Sample Input Sample Output

```
0

Document:
Updated Document:
```

Time Limit: - ms Memory Limit: - kb Code Size: - kb
Q5.
Problem Statement

Kathir wants to create a program that allows him to build a linked list by inserting nodes at the beginning. He wants to be able to input the data for each node and specify when to stop inserting nodes (defined as 0). After inserting the nodes, he wants to display the contents of the linked list. Help Kathir by providing the required input and output formats for the code.

Note: This is a sample question asked in a Wipro interview.

Input Format
The first input consists of an integer, which is the element inserted in the node (n).

The second input consists of the following: if choice 0 is entered, continue to insert the element in the node; otherwise, end the node inserted.

Output Format
For each node inserted at the beginning, the output displays the message "Node inserted" on a new line.

After inserting all the nodes, the output displays the contents of the linked list in a space-separated format on a new line, preceded by the text "Linked List: ".

Finally, the output displays the message "Node ended" on a new line.

Refer to the sample output for the formatting specifications.
Constraints
Choice = 0

1<=n<=100000

Sample Input Sample Output

```
2
0
4
1

Node inserted
Node inserted
Linked List: 4 2
Node ended
```

Sample Input Sample Output

```
4
1

Node inserted
Linked List: 4
Node ended
```

Time Limit: - ms Memory Limit: - kb Code Size: - kb
Q6.
Problem statement

Uma wants to create a program that allows her to build a linked list by inserting nodes at the end. She wants to be able to input the data for each node and specify when to stop inserting nodes (a negative value is entered, indicating the end of the input). After inserting the nodes, she wants to display the contents of the linked list.

Note: This is a sample question asked in a Wipro interview.

Input Format
The input consists of an integer value for each node to be inserted at the end of the linked list.

After inserting each node, enter a non-negative integer indicating the value of the next node to be inserted. If a negative integer is entered, it indicates the end of node insertion.

The input terminates when a negative integer is entered.

Output Format
If the linked list is empty, the output displays the message "Linked List is empty." on a new line.

If the linked list is not empty, the output displays the contents of the linked list in a space-separated format on a new line, preceded by the text "Linked List: ".

Each node's data is displayed in the order it was inserted.

Refer to the sample output for format specifications.

Constraints

The elements of the linked list are integers.

The input will be a series of integers, terminated by a negative value.

The input values can be positive or zero.

Sample Input Sample Output

```
1
2
3
4
-1
```

```
Linked List: 1 2 3 4
```

Sample Input Sample Output

```
-2
```

```
Linked List is empty.
```

Sample Input Sample Output

```
2000
-3000
4000
-5000
-1
```

```
Linked List: 2000
```

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q7.

Problem Statement

Jaanu wants to create a program that allows her to create a linked list of strings. Jaanu to enter the number of strings she wants to insert into the linked list. Then, she should enter each string one by one and insert them at the end of the linked list. After that, ask Jaanu to enter a new string, which will be appended at the end of the linked list. Finally, display the contents of the linked list.

Note: This is a sample question asked in a TCS interview.

Input Format

The first line contains an integer, num_of_strings, representing the number of strings in the linked list.

The next num_of_strings lines contain the strings that constitute the linked list.

The last line contains a string, new_string, which needs to be appended at the end of the linked list.

Output Format

The insert and append at the end of the linked lists after inserting the new node are in the format "Linked List Contents: <list values separated by space>".

Refer to the sample output for format specifications.
Constraints
The number of strings should be a positive integer.

The length of each string should be less than or equal to 100 characters.

Sample Input Sample Output

```
3
Apple
Banana
Orange
Grapes
```

```
Linked List Contents: Apple Banana Orange Grapes
```

Sample Input Sample Output

```
2
Hello
World
Space
```

```
Linked List Contents: Hello World Space
```

Sample Input Sample Output

```
0
None
None
```

```
Linked List Contents: None
```

Time Limit: - ms Memory Limit: - kb Code Size: - kb
Q8.
Problem Statement

Kamal wants to create a linked list and perform the following operations on it:

- Insert a node at the beginning of the linked list.
- Append a node at the end of the linked list.
- Print the final linked list.

Write a program that takes the number of nodes to be inserted, followed by their values, as input. After inserting the nodes, the program should ask for a new value and append a node with that value at the end of the linked list. Finally, the program should print the contents of the linked list.

Example

Input:

5

1 2 3 4 5

6

Output:

Created Linked list: 5 4 3 2 1

Final list: 5 4 3 2 1 6

Explanation:

The program first creates a linked list by inserting nodes at the beginning. The input specifies that there are 5 nodes, and their values are 1, 2, 3, 4, and 5. After inserting these nodes, the program asks for a new value, which is 6. It then appends a node with the value 6 at the end of the linked list. Finally, the program prints the contents of the linked list, which are 5, 4, 3, 2, 1, and 6 in that order.

Note: This is a sample question asked in Capgemini recruitment.
Input Format
The first line consists of an integer, **n**, representing the number of nodes to be initially inserted into the linked list.

The second line of input consists of **n** space-separated integers, representing the elements of the linked list.

The third line of input consists of an integer, **m**, representing the value of the new node to be inserted at the end of the linked list.

Output Format
The first line of output prints the initial linked list elements, separated by space.

The second line of output prints the final linked list elements, after inserting the new node, separated by space.

Refer to the sample output for formatting specifications.

Constraints
The linked list can contain up to 10^2 nodes.

The data for each node is an integer.

Sample Input Sample Output

```
5
1 2 3 4 5
6
```

```
Created Linked list: 5 4 3 2 1
Final list: 5 4 3 2 1 6
```

Sample Input Sample Output

```
3
10 20 30
40
```

```
Created Linked list: 30 20 10
Final list: 30 20 10 40
```

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q9.

Problem Statement

Dhanush wants to create a program that allows him to build a linked list by appending nodes at the end. He also wants to be able to insert a new node at a specified position in the linked list. Dhanush would like to input the data for each node, specify the position and value for the new node, and view the contents of the linked list.

Note: This is a sample question asked in a Capgemini interview.

Input Format

The first line of input contains an integer num_of_nodes, representing the number of nodes to be appended at the end of the linked list.

The next num_of_nodes lines contain an integer value for each node to be appended.

The next line contains two integers, pos and new_val, representing the position and value of the node to be inserted.

The input values are space-separated.

Output Format

The output consists of a single line containing the values of the linked list nodes after the insertion.

The node values are space-separated.

Refer to the sample output for format specifications.

Constraints

The linked list can contain up to 10^2 nodes.

1 <= pos <= number of nodes in the linked list + 1

-10^2 <= new_val <= 10^2

Sample Input Sample Output

```
5
2 3 9 6 1
3
8
```

```
2 3 8 9 6 1
```

Sample Input Sample Output

```
10
8 3 12 77 30 55 94 28 56 14
8
99
```

```
8 3 12 77 30 55 94 99 28 56 14
```

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q10.
Problem Statement

Lisa wants to create a linked list sorted in ascending order. She wants to insert nodes in such a way that the linked list remains sorted after each insertion.

Write a program that takes the number of nodes to be inserted, followed by their values in non-decreasing order. The program should then ask for a new value and insert a node with that value at the appropriate position to maintain the sorted order.

Finally, the program should print the updated linked list.

Example

Input:

5

1 3 5 7 9

4

Output:

1 3 4 5 7 9

Explanation:

The program first creates a sorted linked list using the given input values: 1, 3, 5, 7, and 9. After creating the initial sorted linked list, the program asks for a new value, which is 4. It then inserts a node with value 4 at the appropriate position to maintain the sorted order. Finally, the program prints the updated linked list, which is 1, 3, 4, 5, 7, and 9 in ascending order.

Note: This is a sample question asked in Wipro recruitment.

Input Format
The first line of input consists of an integer **n**, representing the number of elements in the initial sorted linked list.

The second line of input consists of **n** space-separated integers, representing the elements of the sorted linked list.

The third line of input consists of integer **data**, which represents the element to be inserted into the linked list.

Output Format
The program should output the updated linked list after inserting the new element, separated by space.

The linked list should remain sorted in ascending order.

Constraints
The input linked-list is sorted in ascending order.

The linked list can be empty initially.

Sample Input Sample Output

```
5
1 3 5 7 9
4
```

```
1 3 4 5 7 9
```

Sample Input Sample Output

```
6
25 36 47 58 -69 80
-19
```

```
-69 -19 25 36 47 58 80
```

Time Limit: - ms Memory Limit: - kb Code Size: - kb
Q11.
Problem Statement

Vijay wants to create a program that allows him to manipulate a linked list. He wants to be able to perform the following operations:

1: **Append**: Append a node at the end of the linked list.

2: **Append Left:** Append a node at the beginning (left) of the linked list.

3: **Insert:** Insert a node at a specified position in the linked list.

4: **Insert At Key:** Insert a node after a specific key value in the linked list.

5: **Print:** Print the contents of the linked list.

6: **Exit:** Exit the program.

7: **Append Right:** Append a node at the end of the linked list (alternative method).

Note: This is a sample question asked in an Accenture interview.

Input Format
The input consists of multiple lines. Each line represents a single operation to perform on the linked list.

<Choice> is an integer representing the operation to perform.

<Additional Inputs> are the additional inputs required for the chosen operation.

The possible choices are:

Append: <Choice> <Value>

Append Left: <Choice> <Value>

Append Right: <Choice> <Value>

Insert: <Choice> <Position> <Value>

Insert At Key: <Choice> <Key> <Value>

Print: <Choice>

Exit: <Choice>
Output Format
The program produces the following outputs:

For choices 1, 2, 3, 4, and 7, there is no output.

For choice 5 (Print), The elements of the linked list are displayed.

For Choice 6 (Exit): The program terminates.

For an invalid choice, an "Invalid choice" message is displayed.

For the invalid position in the Insert operation, the "Invalid position" message is displayed.

For the key not found in the Insert At Key operation, the "Key not found" message is displayed.

The linked list elements are separated by a space.

Constraints
The input values for the linked list nodes will be non-negative integers.

The position values for insertion will be positive integers.

The key values for insertion after a specific key will be non-negative integers.

The linked list can contain duplicate values.

Sample Input Sample Output

```
1 10
2 20
7 90
5
6
```

```
Linked List: 20 10 90
```

Sample Input Sample Output

```
1 10
1 20
3 2 30
```

```
5
6

Linked List: 10 30 20
```

Sample Input Sample Output

```
1 40
2 20
4 30 40
5
6

Key not found
Linked List: 20 40
```

Sample Input Sample Output

```
1 10
2 20
3 6 30
5
6

Invalid position
Linked List: 20 10
```

Sample Input Sample Output

```
8
5
6

Invalid choice
Linked List:
```

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q12.

Problem Statement

Suresh wants to create a program that allows him to create a linked list of integers. Suresh should enter the number of elements he wants to add to the linked list. Then, he should enter each element one by one and add them to the linked list. Finally, display the contents of the linked list. Suresh should note that the code handles an empty list case and prints a message if the list is empty.

Note: This is a sample question asked in an Accenture interview.

Input Format

The first line of input contains an integer numElements, representing the number of elements to be added to the linked list.

The next numElements lines contain an integer each, representing the elements to be added to the linked list.

Output Format

The output displays the elements of the linked list, separated by a space.

If the linked list is empty, the program will output "The list is empty."

Refer to the sample output for formatting specifications.

Constraints

1<=numElements<=20

Sample Input Sample Output

```
4
7 8 6 4
```

```
7 8 6 4
```

Sample Input Sample Output

```
0
```

```
The list is empty.
```

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Answer Key & Solution

Section 1 - Coding

Q1 Test Case Input Output

```
4
Liam
Emily
James
Mia
Ella 7
```

```
Current Linked List:
Liam Emily James Mia
Updated Linked List:
Liam Emily James Mia Ella
```

Weightage - 10 Input Output

```
2
Jack
Olivia
Sophia 1
```

```
Current Linked List:
Jack Olivia
Updated Linked List:
Sophia Jack Olivia
```

Weightage - 10 Input Output

```
6
John
Alice
Bob
Emma
Michael
Olivia 5
```

```
Current Linked List:
John Alice Bob Emma Michael Olivia
Updated Linked List:
John 5 Alice Bob Emma Michael Olivia
```

Weightage - 15 Input Output

```
3
Henry
Sophia
Ethan
Ava 10
```

```
Current Linked List:
Henry Sophia Ethan
Updated Linked List:
Henry Sophia Ethan Ava
```

Weightage - 15 Input Output

```
7
John
Alice
Bob
Michael
Olivia
Sophia
Ethan
Ava 5
```

```
Current Linked List:
John Alice Bob Michael Olivia Sophia Ethan
Updated Linked List:
John Alice Bob Michael Ava Olivia Sophia Ethan
```

Weightage - 25 Input Output

```
9
Liam
Emma
Daniel
Sophia
Oliver
Charlotte
Mia
Henry
Ella
William 12
```

```
Current Linked List:
Liam Emma Daniel Sophia Oliver Charlotte Mia Henry Ella
Updated Linked List:
Liam Emma Daniel Sophia Oliver Charlotte Mia Henry Ella William
```

Weightage - 25 Sample Input Sample Output

```
4
John
Alice
Bob
Emma
Michael 3
```

```
Current Linked List:
John Alice Bob Emma
Updated Linked List:
John Alice Michael Bob Emma
```

Sample Input Sample Output

```
4
Emma
Daniel
Sophia
Oliver
Charlotte 1
```

```
Current Linked List:
Emma Daniel Sophia Oliver
Updated Linked List:
Charlotte Emma Daniel Sophia Oliver
```

Sample Input Sample Output

```
0
Emma 1
```

```
Current Linked List:

Updated Linked List:
Emma
```

Solution

```cpp
#include <iostream>
#include <string>

using namespace std;

// Define the node structure
struct Node {
    string data;
    Node* next;
};

// Function to create a new node
Node* createNode(const string& data) {
    Node* newNode = new Node;
    newNode->data = data;
    newNode->next = nullptr;
    return newNode;
}

// Function to insert a node at the specified position in the linked list
void insertNode(Node** head, const string& data, int pos) {
    Node* newNode = createNode(data);

    if (pos == 1) {
        // If the position is 1, make the new node the new head
        newNode->next = *head;
        *head = newNode;
    } else {
        Node* temp = *head;

        // Traverse to the node just before the insertion position
        for (int i = 1; i < pos - 1 && temp != nullptr; i++) {
            temp = temp->next;
        }

        if (temp == nullptr) {
            // Invalid position, append the new node at the end of the list
            Node* lastNode = *head;
            while (lastNode->next != nullptr) {
                lastNode = lastNode->next;
            }
            lastNode->next = newNode;
        } else {
            // Insert the new node into the list
            newNode->next = temp->next;
            temp->next = newNode;
        }
    }
}

// Function to display the linked list
void displayList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    int n, pos;
    string data;
    Node* head = nullptr;

    cin >> n;
    for (int i = 1; i <= n; i++) {
        cin >> data;
        insertNode(&head, data, i);
    }

    cout << "Current Linked List:" << endl;
    displayList(head);

    cin >> data >> pos;

    insertNode(&head, data, pos);

    cout << "Updated Linked List:" << endl;
    displayList(head);

    return 0;
}
```

Q2 Test Case Input Output

```
4
Red Green Blue Yellow
Orange 5
```

```
Current Linked List:
Red Green Blue Yellow
Updated Linked List:
Red Green Blue Yellow Orange
```

Weightage - 10 Input Output

```
5
Apple Banana Cherry Durian Elderberry
Fig
3
```

```
Current Linked List:
Apple Banana Cherry Durian Elderberry
Updated Linked List:
Apple Banana Fig Cherry Durian Elderberry
```

Weightage - 10 Input Output

```
7
Dog Cat Bird Fish Monkey Elephant Tiger
Lion 4
```

```
Current Linked List:
Dog Cat Bird Fish Monkey Elephant Tiger
Updated Linked List:
Dog Cat Bird Lion Fish Monkey Elephant Tiger
```

Weightage - 15 Input Output

```
6
Apple Banana Orange Pineapple Mango Grapes
Watermelon 10
```

```
Current Linked List:
Apple Banana Orange Pineapple Mango Grapes
Updated Linked List:
Apple Banana Orange Pineapple Mango Grapes Watermelon
```

Weightage - 15 Input Output

```
8
Apple Banana Cherry Durian Elderberry Fig Grapefruit Hawthorn
Pomegranate 5
```

```
Current Linked List:
Apple Banana Cherry Durian Elderberry Fig Grapefruit Hawthorn
Updated Linked List:
Apple Banana Cherry Durian Pomegranate Elderberry Fig Grapefruit Hawthorn
```

Weightage - 25 Input Output

```
10
One Two Three Four Five Six Seven Eight Nine Ten
Eleven 1
```

```
Current Linked List:
One Two Three Four Five Six Seven Eight Nine Ten
Updated Linked List:
Eleven One Two Three Four Five Six Seven Eight Nine Ten
```

Weightage - 25 Sample Input Sample Output

```
6
A B C D E F
G 7
```

```
Current Linked List:
A B C D E F
Updated Linked List:
A B C D E F G
```

Sample Input Sample Output

```
3
Apple Orange Banana
Mango 1
```

```
Current Linked List:
Apple Orange Banana
Updated Linked List:
Mango Apple Orange Banana
```

Sample Input Sample Output

```
0

Watermelon 1
```

```
Current Linked List:

Updated Linked List:
Watermelon
```

Sample Input Sample Output

```
5
Dog Cat Bird Fish Monkey
Lion 6
```

```
Current Linked List:
Dog Cat Bird Fish Monkey
Updated Linked List:
Dog Cat Bird Fish Monkey Lion
```

Solution

```
#include <iostream>
#include <string>

using namespace std;

// Define the node structure
struct Node {
    string data;
    Node* next;
};

// Function to create a new node
Node* createNode(const string& data) {
    Node* newNode = new Node;
    newNode->data = data;
    newNode->next = nullptr;
    return newNode;
}

// Function to insert a node at the specified position in the linked list
void insertNode(Node** head, const string& data, int pos) {
    Node* newNode = createNode(data);

    if (pos == 1) {
        // If the position is 1, make the new node the new head
        newNode->next = *head;
        *head = newNode;
    } else {
        Node* temp = *head;

        // Traverse to the node just before the insertion position
        for (int i = 1; i < pos - 1 && temp != nullptr; i++) {
            temp = temp->next;
        }

        if (temp == nullptr) {
            // Invalid position, append the new node at the end of the list
            Node* lastNode = *head;
            while (lastNode->next != nullptr) {
                lastNode = lastNode->next;
            }
            lastNode->next = newNode;
        } else {
            // Insert the new node into the list
            newNode->next = temp->next;
            temp->next = newNode;
        }
    }
}

// Function to display the linked list
void displayList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    int n, pos;
    string data;
    Node* head = nullptr;

    cin >> n;
    for (int i = 1; i <= n; i++) {
        cin >> data;
        insertNode(&head, data, i);
    }

    cout << "Current Linked List:" << endl;
    displayList(head);

    cin >> data >> pos;

    insertNode(&head, data, pos);

    cout << "Updated Linked List:" << endl;
    displayList(head);

    return 0;
}
```

Q3 Test Case Input Output

```
2
50 60
70
```

```
Created Linked list: 60 50
Final List: 70 60 50
```

Weightage - 10 Input Output

```
4
8 6 4 2
10
```

```
Created Linked list: 2 4 6 8
Final List: 10 2 4 6 8
```

Weightage - 10 Input Output

```
5
9 18 27 36 45
54
```

```
Created Linked list: 45 36 27 18 9
Final List: 54 45 36 27 18 9
```

Weightage - 15 Input Output

```
6
7 14 21 28 35 42
49
```

```
Created Linked list: 42 35 28 21 14 7
Final List: 49 42 35 28 21 14 7
```

Weightage - 15 Input Output

```
7
3 6 9 12 15 18 21
24
```

```
Created Linked list: 21 18 15 12 9 6 3
Final List: 24 21 18 15 12 9 6 3
```

Weightage - 25 Input Output

```
8
4 8 12 16 20 24 28 32
36
```

```
Created Linked list: 32 28 24 20 16 12 8 4
Final List: 36 32 28 24 20 16 12 8 4
```

Weightage - 25 Sample Input Sample Output

```
5
6 5 4 3 2
1
```

```
Created Linked list: 2 3 4 5 6
Final List: 1 2 3 4 5 6
```

Sample Input Sample Output

```
4
-10 20 -30 40
```

```
5

Created Linked list: 40 -30 20 -10
Final List: 5 40 -30 20 -10
```

Solution

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

// A linked list node
struct Node {
    int data;
    Node* next;
};

// Given a reference (pointer to pointer) to the head of a list and an int, inserts a new node on the front
of the list.
void insertAtFront(Node** head_ref, int new_data)
{
    // 1. Allocate node
    Node* new_node = new Node();

    // 2. Put in the data
    new_node->data = new_data;

    // 3. Make next of new node as head
    new_node->next = (*head_ref);

    // 4. Move the head to point to the new node
    (*head_ref) = new_node;
}

// Function to insert element in LL
void push(Node** head_ref, int new_data)
{
    Node* new_node = new Node();
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

// This function prints contents of linked list starting from head
void printList(Node* node)
{
    while (node != NULL) {
        cout << node->data << " ";
        node = node->next;
    }
    cout << "\n";
}

int main()
{
    // Start with the empty list
    Node* head = NULL;

    int n;
    cin >> n;

    for (int i = 0; i < n; i++) {
        int data;
        cin >> data;
        push(&head, data);
    }

    cout << "Created Linked list: ";
    printList(head);

    // Insert element at the beginning
    int newData;
    cin >> newData;
    insertAtFront(&head, newData);
    cout <<"Final List: " ;
    printList(head);

    return 0;
}
```

Q4 Test Case Input Output

```
4
One
Two
Three
Four
End
```

```
Document: Four Three Two One
Updated Document: Four Three Two One End
```

Weightage - 10 Input Output

```
5
Red
Green
Blue
Yellow
Purple
Rainbow
```

```
Document: Purple Yellow Blue Green Red
Updated Document: Purple Yellow Blue Green Red Rainbow
```

Weightage - 10 Input Output

```
7
Earth
Mars
Venus
Saturn
Jupiter
Neptune
Pluto
Uranus
```

```
Document: Pluto Neptune Jupiter Saturn Venus Mars Earth
Updated Document: Pluto Neptune Jupiter Saturn Venus Mars Earth Uranus
```

Weightage - 25 Input Output

```
5
Red
Green
Blue
Yellow
Orange
Purple
```

```
Document: Orange Yellow Blue Green Red
Updated Document: Orange Yellow Blue Green Red Purple
```

Weightage - 15 Input Output

```
6
January
February
March
April
May
June
December
```

```
Document: June May April March February January
Updated Document: June May April March February January December
```

Weightage - 15 Input Output

```
6
Alpha
Beta
Gamma
Delta
Epsilon
Zeta
Iota
```

```
Document: Zeta Epsilon Delta Gamma Beta Alpha
Updated Document: Zeta Epsilon Delta Gamma Beta Alpha Iota
```

Weightage - 25 Sample Input Sample Output

```
3
Apple
Banana
Orange
Grapes
```

```
Document: Orange Banana Apple
Updated Document: Orange Banana Apple Grapes
```

Sample Input Sample Output

```
2
Hello
World
Space
```

```
Document: World Hello
Updated Document: World Hello Space
```

Sample Input Sample Output

```
0
```

```
Document:
Updated Document:
```

Solution

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

// A linked list node
struct Node {
    string data;
    Node* next;
};

// Given a reference (pointer to pointer) to the head of a list and a string, inserts a new node at the front
of the list.
void push(Node** head_ref, string new_data)
{
    // Create a new node
    Node* new_node = new Node();
    new_node->data = new_data;

    // Make the new node point to the current head
    new_node->next = (*head_ref);

    // Update the head to point to the new node
    (*head_ref) = new_node;
}

// Given a reference (pointer to pointer) to the head of a list and a string, appends a new node at the end
void append(Node** head_ref, string new_data)
{
    // Create a new node
    Node* new_node = new Node();
    new_node->data = new_data;

    // Store the head reference in a temporary variable
    Node* last = *head_ref;

    // Set the next pointer of the new node as NULL since it will be the last node
    new_node->next = NULL;

    // If the Linked List is empty, make the new node as the head and return
    if (*head_ref == NULL) {
        *head_ref = new_node;
        return;
    }

    // Else traverse till the last node
    while (last->next != NULL) {
        last = last->next;
    }

    // Change the next pointer of the last node to point to the new node
    last->next = new_node;
}

// This function prints the contents of the linked list starting from the head
void printList(Node* node)
{
    while (node != NULL) {
        cout << " " << node->data;
        node = node->next;
    }
}

int main()
{
    // Start with an empty list
    Node* head = NULL;
    int n;
    cin >> n;

    for (int i = 0; i < n; i++) {
        string str;
        cin >> str;
        push(&head, str);
    }

    cout << "Document:";
    printList(head);

    string new_string;
    cin >> new_string;

    // Insert the new string at the end
    append(&head, new_string);

    cout << "\nUpdated Document:";
    printList(head);

    return 0;
}
```

Q5 Test Case Input Output

```
5
0
6
0
1
8
```

```
Node inserted
Node inserted
Node inserted
Linked List: 1 6 5
Node ended
```

Weightage - 10 Input Output

```
3
1
```

```
Node inserted
Linked List: 3
Node ended
```

Weightage - 10 Input Output

```
5
0
7
0
2
0
4
0
1
3
```

```
Node inserted
Node inserted
Node inserted
Node inserted
Node inserted
Linked List: 1 4 2 7 5
Node ended
```

Weightage - 15 Input Output

```
3
0
4
0
4
5
```

```
Node inserted
Node inserted
Node inserted
Linked List: 4 4 3
Node ended
```

Weightage - 15 Input Output

```
5
0
20
0
890
0
3
0
43
0
245
0
542
1
```

```
Node inserted
Node inserted
Node inserted
Node inserted
Node inserted
Node inserted
Node inserted
Linked List: 542 245 43 3 890 20 5
Node ended
```

Weightage - 25 Input Output

```
6
0
3
0
9
0
234
0
513
0
3412
0
4523
0
134124
```

```
0
13
1

Node inserted
Node inserted
Node inserted
Node inserted
Node inserted
Node inserted
Node inserted
Node inserted
Node inserted
Linked List: 13 134124 4523 3412 513 234 9 3 6
Node ended
```

Weightage - 25 Sample Input Sample Output

```
2
0
4
1

Node inserted
Node inserted
Linked List: 4 2
Node ended
```

Sample Input Sample Output

```
4
1

Node inserted
Linked List: 4
Node ended
```

Solution

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

struct Node {
    int data;
    Node* next;
};

Node* head = nullptr;

void insertAtBeginning(int item) {
    Node* newNode = new Node;
    newNode->data = item;
    newNode->next = head;
    head = newNode;
    cout << "Node inserted" << endl;
}

void traverseLinkedList() {
    Node* current = head;
    cout << "Linked List: ";
    while (current != nullptr) {
        cout << current->data << " ";
        current = current->next;
    }
    cout << endl;
}

int main() {
    int choice, item;

    do {
        cin >> item;
        insertAtBeginning(item);
        cin >> choice;
    } while (choice == 0);

    traverseLinkedList();

    cout << "Node ended";
    return 0;
}
```

Q6 Test Case Input Output

```
53
2
234
243
14
-4
```

```
Linked List: 53 2 234 243 14
```

Weightage - 15 Input Output

```
57
349
1430
1234
134
13
-5
```

```
Linked List: 57 349 1430 1234 134 13
```

Weightage - 15 Input Output

```
8
3
1
2
3
4
5
-6
```

```
Linked List: 8 3 1 2 3 4 5
```

Weightage - 10 Input Output

```
-90
```

```
Linked List is empty.
```

Weightage - 10 Input Output

```
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
-1
```

```
Linked List: 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200
```

Weightage - 25 Input Output

```
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
-1
```

```
Linked List: 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75
```

Weightage - 25 Sample Input Sample Output

```
1
2
3
4
-1
```

```
Linked List: 1 2 3 4
```

Sample Input Sample Output

```
-2
```

```
Linked List is empty.
```

Sample Input Sample Output

```
2000
-3000
4000
-5000
-1
```

```
Linked List: 2000
```

Solution

```c
#include<stdio.h>
#include<stdlib.h>

struct Node
{
    int data;
    struct Node *next;
};

void print(struct Node *head)
{
    struct Node *node = head;
    while(node != NULL)
    {
        printf("%d\n", node->data);
        node = node->next;
    }
}

void insert(struct Node **head, int value)
{
    struct Node *last = *head;
    struct Node *newnode = (struct Node*) malloc (sizeof(struct Node));
    newnode->data = value;
    newnode->next = NULL;

    if(*head == NULL)
    {
        *head = newnode;
    }
    else
    {
        while(last->next != NULL)
        {
            last = last->next;
        }
        last->next = newnode;
    }
}

int main()
{
    struct Node *head = NULL;
    int elements;
    while(1)
    {
        scanf("%d", &elements);
        if(elements >= 0)
        {
            insert(&head, elements);
        }
        else
        {
            break;
        }
    }

    print(head);
}
```

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

struct Node
{
    int data;
    struct Node *next;
};

void print(struct Node *head)
{
    if (head == NULL)
    {
        cout << "Linked List is empty." << endl;
        return;
    }

    struct Node *current = head;
    cout << "Linked List: ";
    while (current != NULL)
    {
        cout << current->data << " ";
        current = current->next;
    }
    cout << endl;
}

void insert(struct Node **head, int value)
{
    Node *last = *head;
    Node *newnode = new Node;
    newnode->data = value;
    newnode->next = NULL;

    if(*head == NULL)
    {
        *head = newnode;
    }
    else
    {
        while(last->next != NULL)
        {
            last = last->next;
        }
        last->next = newnode;
    }
}

int main()
{
    struct Node *head = NULL;
    int elements;
    while(1)
    {
        cin >> elements;
        if(elements >= 0)
        {
            insert(&head, elements);
        }
        else
        {
            break;
        }
    }
```

```
        print(head);
    }
```

Q7 Test Case Input Output

```
4
One
Two
Three
Four
End
```

```
Linked List Contents: One Two Three Four End
```

Weightage - 10 Input Output

```
5
Red
Green
Blue
Yellow
Purple
Rainbow
```

```
Linked List Contents: Red Green Blue Yellow Purple Rainbow
```

Weightage - 10 Input Output

```
7
Earth
Mars
Venus
Saturn
Jupiter
Neptune
Pluto
Uranus
```

```
Linked List Contents: Earth Mars Venus Saturn Jupiter Neptune Pluto Uranus
```

Weightage - 25 Input Output

```
5
Red
Green
Blue
Yellow
Orange
Purple
```

```
Linked List Contents: Red Green Blue Yellow Orange Purple
```

Weightage - 15 Input Output

```
6
January
February
March
April
May
June
December
```

```
Linked List Contents: January February March April May June December
```

Weightage - 15 Input Output

```
6
Alpha
Beta
```

```
Gamma
Delta
Epsilon
Zeta
Iota

Linked List Contents: Alpha Beta Gamma Delta Epsilon Zeta Iota
```

Weightage - 25 Sample Input Sample Output

```
3
Apple
Banana
Orange
Grapes

Linked List Contents: Apple Banana Orange Grapes
```

Sample Input Sample Output

```
2
Hello
World
Space

Linked List Contents: Hello World Space
```

Sample Input Sample Output

```
0
None
None

Linked List Contents: None
```

Solution

```
#include <iostream>
#include <string>

using namespace std;

struct Node {
    string data;
    Node* next;
};

void insertAtEnd(Node** head, string newData) {
    Node* newNode = new Node();
    newNode->data = newData;
    newNode->next = nullptr;

    if (*head == nullptr) {
        *head = newNode;
    } else {
        Node* current = *head;
        while (current->next != nullptr) {
            current = current->next;
        }
        current->next = newNode;
    }
}

void appendToEnd(Node** head, string newString) {
    insertAtEnd(head, newString);
}

void printLinkedList(Node* head) {
    Node* current = head;
    while (current != nullptr) {
        cout << current->data << " ";
        current = current->next;
    }
    cout << endl;
}

int main() {
    Node* head = nullptr;
    int numStrings;

    cin >> numStrings;

    for (int i = 0; i < numStrings; i++) {
        string str;
        cin >> str;
        insertAtEnd(&head, str);
    }

    string newString;
    cin >> newString;
    appendToEnd(&head, newString);

    cout << "Linked List Contents: ";
    printLinkedList(head);

    Node* current = head;
    while (current != nullptr) {
        Node* temp = current;
        current = current->next;
        delete temp;
    }
    return 0;
}
```

Q8 Test Case Input Output

```
4
2 4 6 8
10
```

```
Created Linked list: 8 6 4 2
Final list: 8 6 4 2 10
```

Weightage - 10 Input Output

```
6
7 14 21 28 35 42
49
```

```
Created Linked list: 42 35 28 21 14 7
Final list: 42 35 28 21 14 7 49
```

Weightage - 10 Input Output

```
8
3 6 9 12 15 18 21 24
27
```

```
Created Linked list: 24 21 18 15 12 9 6 3
Final list: 24 21 18 15 12 9 6 3 27
```

Weightage - 15 Input Output

```
12
2 4 6 8 10 12 14 16 18 20 22 24
26
```

```
Created Linked list: 24 22 20 18 16 14 12 10 8 6 4 2
Final list: 24 22 20 18 16 14 12 10 8 6 4 2 26
```

Weightage - 25 Input Output

```
7
2 4 6 8 10 12 14
16
```

```
Created Linked list: 14 12 10 8 6 4 2
Final list: 14 12 10 8 6 4 2 16
```

Weightage - 15 Input Output

```
10
5 10 15 20 25 30 35 40 45 50
55
```

```
Created Linked list: 50 45 40 35 30 25 20 15 10 5
Final list: 50 45 40 35 30 25 20 15 10 5 55
```

Weightage - 25 Sample Input Sample Output

```
5
1 2 3 4 5
6
```

```
Created Linked list: 5 4 3 2 1
Final list: 5 4 3 2 1 6
```

Sample Input Sample Output

```
3
10 20 30
```

```
40

Created Linked list: 30 20 10
Final list: 30 20 10 40
```

Solution

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

struct Node {
    int data;
    Node* next;
};

void push(Node** head_ref, int new_data)
{
    Node* new_node = new Node();
    new_node->data = new_data;

    new_node->next = (*head_ref);

    (*head_ref) = new_node;
}


void append(Node** head_ref, int new_data)
{
    Node* new_node = new Node();
    new_node->data = new_data;

    Node* last = *head_ref;

    new_node->next = NULL;

    if (*head_ref == NULL) {
        *head_ref = new_node;
        return;
    }

    while (last->next != NULL) {
        last = last->next;
    }

    last->next = new_node;
}

void printList(Node* node)
{
    while (node != NULL) {
        cout << " " << node->data;
        node = node->next;
    }
}

int main()
{
    Node* head = NULL;

    int num_of_nodes, new_val;
    cin >> num_of_nodes;

    for (int i = 0; i < num_of_nodes; i++) {
        int val;
        cin >> val;
        push(&head, val);
    }

    cout << "Created Linked list:";
    printList(head);

    cin >> new_val;
    append(&head, new_val);

    cout << "\nFinal list:";
    printList(head);

    return 0;
}
```

Q9 Test Case Input Output

```
15
285 265 402 395 179 250 500 488 327 124 427 164 219 176 192
10
72
```

```
285 265 402 395 179 250 500 488 327 72 124 427 164 219 176 192
```

Weightage - 25 Input Output

```
5
340 322 224 427 399
2
100
```

```
340 100 322 224 427 399
```

Weightage - 10 Input Output

```
10
405 542 743 391 583 713 185 93 881 491
8
49
```

```
405 542 743 391 583 713 185 49 93 881 491
```

Weightage - 20 Input Output

```
15
88 65 72 183 31 414 83 434 370 161 336 260 480 107 23
14
49
```

```
88 65 72 183 31 414 83 434 370 161 336 260 480 49 107 23
```

Weightage - 20 Input Output

```
40
1753 2265 3448 3839 5348 1361 3171 6631 4792 1384 6277 1284 3844 5128 7030 4825 3769 5017 4722 4873 4569 6511
6212 5114 1007 3095 2598 2065 1848 4153 6318 8466 1617 1138 5539 7453 6298 2690 6841 8118
38
2000
```

```
1753 2265 3448 3839 5348 1361 3171 6631 4792 1384 6277 1284 3844 5128 7030 4825 3769 5017 4722 4873 4569 6511
6212 5114 1007 3095 2598 2065 1848 4153 6318 8466 1617 1138 5539 7453 6298 2000 2690 6841 8118
```

Weightage - 25 Sample Input Sample Output

```
5
2 3 9 6 1
3
8
```

```
2 3 8 9 6 1
```

Sample Input Sample Output

```
10
8 3 12 77 30 55 94 28 56 14
8
99
```

```
8 3 12 77 30 55 94 99 28 56 14
```

Solution

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct myNode {
    int val;
    struct myNode * next;
}
Node;
void append(Node ** head, int val) {
    Node * tmp = (Node * ) malloc(sizeof(Node));
    tmp -> val = val;
    tmp -> next = NULL;
    if ( * head == NULL) {
        * head = tmp;
    } else {
        Node * curr = * head;
        while (curr -> next != NULL) {
            curr = curr -> next;
        }
        curr -> next = tmp;
    }
}
void print(Node * head) {
    if (head == NULL)
        return;
    Node * curr = head;
    printf("%d ", curr -> val);
    print(curr -> next);
}
void insert(Node ** head, int pos, int new_val) {
    Node * tmp = (Node * ) malloc(sizeof(Node));
    tmp -> val = new_val;
    tmp -> next = NULL;
    int i = 1;
    Node * curr = * head;
    Node * last = NULL;
    while (i < pos) {
        last = curr;
        curr = curr -> next;
        i++;
    }
    last -> next = tmp;
    tmp -> next = curr;
}

int main(void) {
    int num_of_nodes, i, pos, new_val;
    scanf("%d", & num_of_nodes);
    Node * myList = NULL;
    for (i = 0; i < num_of_nodes; i++) {
        int val;
        scanf("%d", & val);
        append( & myList, val);
    }
    scanf("%d", & pos);
    scanf("%d", & new_val);
    insert( & myList, pos, new_val);
    print(myList);
    return 0;
}
```

```
#include <iostream>

struct Node {
    int val;
    Node* next;
};

void append(Node** head, int val) {
    Node* tmp = new Node;
    tmp->val = val;
    tmp->next = nullptr;

    if (*head == nullptr) {
        *head = tmp;
    } else {
        Node* curr = *head;
        while (curr->next != nullptr) {
            curr = curr->next;
        }
        curr->next = tmp;
    }
}

void print(Node* head) {
    if (head == nullptr)
        return;

    Node* curr = head;
    std::cout << curr->val << " ";
    print(curr->next);
}

void insert(Node** head, int pos, int new_val) {
    Node* tmp = new Node;
    tmp->val = new_val;
    tmp->next = nullptr;

    int i = 1;
    Node* curr = *head;
    Node* last = nullptr;

    while (i < pos) {
        last = curr;
        curr = curr->next;
        i++;
    }

    if (last == nullptr) {
        tmp->next = *head;
        *head = tmp;
    } else {
        last->next = tmp;
        tmp->next = curr;
    }
}

int main() {
    int num_of_nodes, pos, new_val;
    std::cin >> num_of_nodes;

    Node* myList = nullptr;

    for (int i = 0; i < num_of_nodes; i++) {
        int val;
        std::cin >> val;
        append(&myList, val);
    }

    std::cin >> pos >> new_val;
    insert(&myList, pos, new_val);

    print(myList);

    return 0;
}
```

Q10 Test Case Input Output

```
5
10 20 30 40 50
25
```

```
10 20 25 30 40 50
```

Weightage - 10 Input Output

```
3
-5 0 10
-2
```

```
-5 -2 0 10
```

Weightage - 10 Input Output

```
4
-10 -5 0 10
-15
```

```
-15 -10 -5 0 10
```

Weightage - 15 Input Output

```
7
-100 -50 0 20 50 100 200
25
```

```
-100 -50 0 20 25 50 100 200
```

Weightage - 15 Input Output

```
9
-40 -30 -20 -10 0 10 20 30 40
5
```

```
-40 -30 -20 -10 0 5 10 20 30 40
```

Weightage - 25 Input Output

```
8
10 20 30 40 50 60 70 80
35
```

```
10 20 30 35 40 50 60 70 80
```

Weightage - 25 Sample Input Sample Output

```
5
1 3 5 7 9
4
```

```
1 3 4 5 7 9
```

Sample Input Sample Output

```
6
25 36 47 58 -69 80
-19
```

```
-69 -19 25 36 47 58 80
```

Solution

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

/* Link list node */
struct Node {
    int data;
    Node* next;
};

/* Function to insert a new_node in a list. Note that this function expects a pointer to head_ref as this can
modify the head of the input linked list (similar to push()) */
void sortedInsert(Node** head_ref, int new_data)
{
    Node* new_node = new Node();
    new_node->data = new_data;
    new_node->next = NULL;

    Node* current;
    /* Special case for the head end */
    if (*head_ref == NULL || (*head_ref)->data >= new_data) {
        new_node->next = *head_ref;
        *head_ref = new_node;
    }
    else {
        /* Locate the node before the point of insertion */
        current = *head_ref;
        while (current->next != NULL && current->next->data < new_data) {
            current = current->next;
        }
        new_node->next = current->next;
        current->next = new_node;
    }
}

/* Function to print linked list */
void printList(Node* head)
{
    Node* temp = head;
    while (temp != NULL) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

/* Driver program to test count function */
int main()
{
    /* Start with the empty list */
    Node* head = NULL;
    int n, data;

    cin >> n;

    for (int i = 0; i < n; i++) {
        cin >> data;
        sortedInsert(&head, data);
    }

    cin >> data;
    sortedInsert(&head, data);

    printList(head);

    return 0;
}
```

Q11 Test Case Input Output

```
1 10
1 20
4 10 30
5
6
```

```
Linked List: 10 30 20
```

Weightage - 10 Input Output

```
1 60
2 10
7 80
5
6
```

```
Linked List: 10 60 80
```

Weightage - 10 Input Output

```
1 10
3 1 20
5
6
```

```
Linked List: 20 10
```

Weightage - 15 Input Output

```
1 10
1 20
2 30
2 40
3 5 50
4 20 60
7 70
3 10 80
4 50 90
4 100 110
5
6
```

```
Invalid position
Key not found
Linked List: 40 30 10 20 60 50 90 70
```

Weightage - 15 Input Output

```
1 10
2 20
3 2 30
4 20 40
4 10 50
7 100
5
6
```

```
Linked List: 20 40 30 10 50 100
```

Weightage - 25 Input Output

```
1 10
1 20
2 30
2 40
```

```
3 5 50
4 20 60
7 70
3 10 80
4 50 90
5
6
```

```
Invalid position
Linked List: 40 30 10 20 60 50 90 70
```

Weightage - 25 Sample Input Sample Output

```
1 10
2 20
7 90
5
6
```

```
Linked List: 20 10 90
```

Sample Input Sample Output

```
1 10
1 20
3 2 30
5
6
```

```
Linked List: 10 30 20
```

Sample Input Sample Output

```
1 40
2 20
4 30 40
5
6
```

```
Key not found
Linked List: 20 40
```

Sample Input Sample Output

```
1 10
2 20
3 6 30
5
6
```

```
Invalid position
Linked List: 20 10
```

Sample Input Sample Output

```
8
5
6
```

```
Invalid choice
Linked List:
```

Solution

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

struct Node {
    int val;
    Node* next;
};

void append(Node** head, int val) {
    Node* newNode = new Node;
    newNode->val = val;
    newNode->next = nullptr;

    if (*head == nullptr) {
        *head = newNode;
    } else {
        Node* current = *head;
        while (current->next != nullptr) {
            current = current->next;
        }
        current->next = newNode;
    }
}

void appendLeft(Node** head, int val) {
    Node* newNode = new Node;
    newNode->val = val;
    newNode->next = *head;
    *head = newNode;
}

void appendRight(Node** head, int val) {
    Node* newNode = new Node;
    newNode->val = val;
    newNode->next = nullptr;

    if (*head == nullptr) {
        *head = newNode;
    } else {
        Node* current = *head;
        while (current->next != nullptr) {
            current = current->next;
        }
        current->next = newNode;
    }
}

void insert(Node** head, int pos, int new_val) {
    if (pos <= 0) {
        cout << "Invalid position" << endl;
        return;
    }

    Node* newNode = new Node;
    newNode->val = new_val;
    newNode->next = nullptr;

    if (pos == 1) {
        newNode->next = *head;
        *head = newNode;
        return;
    }

    int i = 1;
    Node* current = *head;
    Node* prev = nullptr;
    while (i < pos && current != nullptr) {
        prev = current;
        current = current->next;
        i++;
    }

    if (i == pos) {
        prev->next = newNode;
        newNode->next = current;
    } else {
        cout << "Invalid position" << endl;
    }
}

void insertAtKey(Node** head, int key, int new_val) {
    Node* newNode = new Node;
    newNode->val = new_val;
    newNode->next = nullptr;

    if (*head == nullptr) {
        *head = newNode;
        return;
    }

    Node* current = *head;
    while (current != nullptr) {
        if (current->val == key) {
            newNode->next = current->next;
            current->next = newNode;
            return;
        }
        current = current->next;
    }

    cout << "Key not found" << endl;
}

void print(Node* head) {
    Node* current = head;
    while (current != nullptr) {
        cout << current->val;
        current = current->next;
        if (current != nullptr) {
            cout << " ";
        }
    }
}

int main() {
    Node* myList = nullptr;
    int choice;
    int numElements;
    int val;
    int insertPos;
    int insertVal;
    int key;
    int keyVal;

    do {
        cin >> choice;

        switch (choice) {
            case 1:
                cin >> val;
                append(&myList, val);
                break;
            case 2:
                cin >> val;
                appendLeft(&myList, val);
                break;
            case 3:
                cin >> insertPos;
                cin >> insertVal;
                insert(&myList, insertPos, insertVal);
                break;
            case 4:
                cin >> key;
                cin >> keyVal;
                insertAtKey(&myList, key, keyVal);
                break;
            case 5:
                cout << "Linked List: ";
                print(myList);
                break;
            case 6:
                break;
            case 7:
                cin >> val;
                appendRight(&myList, val);
                break;
            default:
                cout << "Invalid choice" << endl;
        }

    } while (choice != 6);

    return 0;
}
```

Q12 Test Case Input Output

```
3
8 5 3

8 5 3
```

Weightage - 10 Input Output

```
6
9 5 4 3 7 1

9 5 4 3 7 1
```

Weightage - 10 Input Output

```
8
7 6 5 4 3 7 2 1

7 6 5 4 3 7 2 1
```

Weightage - 15 Input Output

```
10
5 3 6 2 1 7 8 9 10 32

5 3 6 2 1 7 8 9 10 32
```

Weightage - 15 Input Output

```
15
68 45 78 87 54 23 57 89 90 41 45 60 63 32 31

68 45 78 87 54 23 57 89 90 41 45 60 63 32 31
```

Weightage - 25 Input Output

```
20
7 4 6 3 1 4 6 4 3 6 4 1 7 4 2 6 3 6 9 10
```

```
7 4 6 3 1 4 6 4 3 6 4 1 7 4 2 6 3 6 9 10
```

Weightage - 25 Sample Input Sample Output

```
4
7 8 6 4
```

```
7 8 6 4
```

Sample Input Sample Output

```
0
```

```
The list is empty.
```

Solution

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

struct Node {
    int data;
    Node* next;
};

class LinkedList {
  private:
    Node* head;
  public:
    LinkedList(){
      head = NULL;
    }

    void push_back(int newElement) {
      Node* newNode = new Node();
      newNode->data = newElement;
      newNode->next = NULL;
      if(head == NULL) {
        head = newNode;
      } else {
        Node* temp = head;
        while(temp->next != NULL)
          temp = temp->next;
        temp->next = newNode;
      }
    }

    void PrintList() {
      Node* temp = head;
      if(temp != NULL) {
        while(temp != NULL) {
          cout << temp->data << " ";
          temp = temp->next;
        }
        cout << endl;
      } else {
        cout << "The list is empty.\n";
      }
    }
};

int main() {
  LinkedList MyList;

  int numElements;
  cin >> numElements;

  for (int i = 0; i < numElements; i++) {
    int element;
    cin >> element;
    MyList.push_back(element);
  }

  MyList.PrintList();

  return 0;
}
```