LPU_Colab

DSA_Unit 2_Lecture 10 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 are given an integer **n** representing the number of nodes in a singly linked list. Each node contains a string value.

Your task is to implement a program that creates a singly linked list with the given number of nodes and string values and then **deletes the last node** from the list. Finally, you need to print the contents of the modified linked list.

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 nodes in the singly linked list.

The next **n** lines of input consist of **n** strings, where each line represents the string value of a node in the linked list.

Output Format
The output should print the elements of the singly linked list after deleting the last node. The elements should be separated by space.

Constraints
1 ≤ n≤ 100

Each string value in the nodes will contain only alphabetical characters.

Sample Input Sample Output

```
4
Hello
World
I
Am

Hello World I
```

Sample Input Sample Output

```
3
Hello
Hello
Hello

Hello Hello
```

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

Problem Statement

You are developing a program for an online shopping cart system. Each node in the singly linked list represents a product added to the cart. The program should allow for deleting a product from the cart at a **specific position** and displaying the updated cart.

Write a program to implement the online shopping cart system according to the following specifications:

1. The program should prompt the user to enter the number of products in the cart.
2. For each product, the program should prompt the user to enter its product code (an integer value).
3. The program should create a singly linked list with nodes representing the products in the cart, where each node contains the product code.
4. The program should then prompt the user to enter the position of the product to be deleted from the cart.
5. The program should delete the node at the specified position from the linked list.
6. Finally, the program should traverse the updated linked list and print the product codes of the remaining products in the cart.

Note: This is a sample question asked in a TCS interview.
Input Format
The first line of input consists of an integer **n**, the number of nodes in the linked list.

The second line of input consists of **n** integers, representing the data value of each node in the linked list, separated by space.

The third line of input consists of an integer **x**, representing the position of the node to be deleted. (1-based)

Output Format
The output consists of a single line containing the data values of the modified linked list after deleting the node at position **x**. The data values should be separated by a space.

Constraints
1 ≤ n ≤ 10^5 (number of elements in the linked list)

-10^9 ≤ value of each node ≤ 10^9

1 ≤ x ≤ n (position of the node to be deleted)

Sample Input Sample Output

```
7
7270 5318 2964 4237 7388 7010 2468
4

7270 5318 2964 7388 7010 2468
```

Sample Input Sample Output

```
6
10 20 30 40 50 60
3

10 20 40 50 60
```

Time Limit: - ms Memory Limit: - kb Code Size: - kb
Q3.
Problem Statement

Your task is to write a program that takes input for the number of elements in the linked list and the corresponding string values for each element. Based on this input, your program should create a linked list and then delete the alternate nodes from it.

Note: This is a sample question asked in a HCL interview.
Input Format
The first line contains an integer **n,** the number of elements in the linked list.

The second line contains **n** space-separated strings representing the elements of the linked list.

Output Format
If the linked list is empty, output "List is empty".

If the linked list is not empty, output the following:

The first line should display the elements of the original linked list, separated by a space.

The second line should display the elements of the linked list after deleting the alternate nodes, separated by a space.

Refer to the sample output for formatting specifications.

Constraints
0 ≤ n ≤ 100

Each string in the input is non-empty and consists of alphanumeric characters (letters and digits) only.

The total length of all strings combined does not exceed 10^2 characters.

Sample Input Sample Output

```
2
Apple Banana

Linked list data: Apple Banana
After deleting alternate node:Apple
```

Sample Input Sample Output

```
5
Red Green Blue Yellow Orange

Linked list data: Red Green Blue Yellow Orange
After deleting alternate node:Red Blue Orange
```

Sample Input Sample Output

```
0

List is empty
```

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

You are tasked with developing a program to manage a student records system using a singly linked list. Each node in the linked list represents a student and contains their unique ID. The program should allow for creating a list of student records, **deleting the first student's record**, and printing the remaining student IDs.

Write a program to implement the student records system according to the following specifications:

1. The program should prompt the user to enter the number of students **n** for which records need to be created.
2. For each student, the program should prompt the user to enter their student ID (an integer value).
3. The program should create a singly linked list with **n** nodes, where each node represents a student and contains their student ID.
4. After creating the linked list, the program should delete the first node from the list.
5. Finally, the program should traverse the modified linked list and print the student IDs of the remaining students.

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 nodes in the linked list.

The second line of input consists of **n** space-separated integers, representing the values of each node in the linked list.

Output Format
The output prints the space-separated values of the linked list after deleting the first node.

Constraints
n>0

Each node value is an integer.

Sample Input Sample Output

```
5
2 3 6 9 8

3 6 9 8
```

Sample Input Sample Output

```
20
92 8 14 44 17 55 47 93 22 5 11 34 45 87 80 16 63 15 31 10

8 14 44 17 55 47 93 22 5 11 34 45 87 80 16 63 15 31 10
```

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

Nandha wants to write a program that deletes a range of nodes between two given positions in a singly linked list. Write a program that takes input for the size of the linked list, the elements of the linked list, the start position, and the end position. The program should delete the nodes between the specified positions

and display the updated linked list.

Note: This is a sample question asked in a TCS interview.
Input Format
The first line contains an integer 'size' representing the size of the linked list.

The second line contains 'arr' space-separated integers representing the elements of the linked list.

The third line contains an integer 'start' representing the start position.

The fourth line contains an integer 'end' representing the end position.

Output Format
The first line of output displays the linked list before deletion.

The second line of output displays the linked list after deletion.

Refer to the sample output for formatting specifications.

Constraints
1 <= size <= 100

-50000 <= arr <= 50000

1 <= start <= 100

1 <= end <= 100

Sample Input Sample Output

```
5
1 2 3 4 5
1
3

Linked List before deletion: 1 2 3 4 5
Linked List after deletion: 4 5
```

Sample Input Sample Output

```
5
-50000 50000 4000 3676 7263
1
5

Linked List before deletion: -50000 50000 4000 3676 7263
Linked List after deletion: all the elements are deleted
```

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

Prackya wants to implement a function to delete the middle node of a singly linked list. She wants you to write a program that takes user input to construct a linked list, inserts nodes at the end, and then deletes the middle node if it exists. If the linked list has an even number of nodes, the function should delete the second middle node.

For example, if the given linked list is 1->2->3->4->5 then the linked list should be modified to 1->2->4->5.

For example, if the given linked list is 1->2->3->4->5->6, then it should be modified to 1->2->3->5->6.

Note: This is a sample question asked in an AMCAT interview.
Input Format
The first line of input consists of the number of nodes in the linked list (numNodes)

The second line of input consists of the values for each node, inserted at the end of the list.

Output Format
The output displays the original linked list before deleting the middle node.

The updated linked list after deleting the middle node.

Refer to the sample output for format specifications.

Constraints
1 <= numNodes <= 100

-50000 <= values of nodes <= 50000

Sample Input Sample Output

```
5
1 2 3 4 5

Original Linked List: 1 2 3 4 5
Updated Linked List: 1 2 4 5
```

Sample Input Sample Output

```
6
1 2 3 4 5 6

Original Linked List: 1 2 3 4 5 6
Updated Linked List: 1 2 3 5 6
```

Time Limit: - ms Memory Limit: - kb Code Size: - kb
Q7.
Problem Statement

Elsa wants to delete nodes with even positions in a singly linked list. She needs your help to write a program that takes the size of the linked list and its elements as input and uses insertion at the end to add nodes to the linked list and delete nodes with even positions from the linked list. She wants to see the original linked list before deletion and the final linked list after the deletion process.

Write a program to solve Elsa's problem.

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

Input Format
The first line of the input consists of the size (n) of the linked list (an integer).

The second line of the input consists of the elements of the linked list (a sequence of space-separated integers).

Output Format
The output displays the original linked list before the deletion process.

The final linked list after deleting nodes with even positions.

Refer to the sample output for format specifications.

Constraints
1<=n<=100

-10000<=arr<=10000

Sample Input Sample Output

```
5
100 200 300 10000 9999

Original Linked List: 100 200 300 10000 9999
Final Linked List: 100 300 9999
```

Sample Input Sample Output

```
3
-10000 -2000 -3000

Original Linked List: -10000 -2000 -3000
Final Linked List: -10000 -3000
```

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

Madhev wants to remove all nodes with values greater than a specified value 'x' from a singly linked list. He needs your help to write a program that takes the size of the linked list, the elements of the linked list, and the value 'x' as input.

Additionally, he wants to insert new nodes at the end of the linked list. The program should then delete all nodes with values greater than 'x' from the linked list and display the modified linked list.

Write a program to solve Madhev's problem.

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

Input Format
The first line of input consists of the size of the linked list n (an integer).

The second line of input consists of the elements of the linked list arr (a sequence of space-separated integers).

The last line of input consists of the value 'x' (an integer) to compare against the nodes.
Output Format
The output displays the original linked list.

The modified linked list after removing nodes with values greater than 'x'.

Refer to the sample output for format specifications.

Constraints
1<=n<=1=100

-10000<=arr<=10000

-10000<=x<=10000

Sample Input Sample Output

```
5
8 7 5 3 2
5

Original Linked List: 8 7 5 3 2
Modified Linked List: 5 3 2
```

Sample Input Sample Output

```
5
-8 7 -5 -3 -2
0

Original Linked List: -8 7 -5 -3 -2
Modified Linked List: -8 -5 -3 -2
```

Time Limit: - ms Memory Limit: - kb Code Size: - kb
Q9.
Problem Statement

Vennila is a student. She is learning data structure and a singly linked list. She wants to write a program to delete the second-to-last node of the linked list and also implement a program that deletes the second-to-last node of a singly linked list. Define a struct Node with two members: data to store the integer value and next to store the pointer to the next node in the list.

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

Input Format
The input consists of the following:

The first line contains an integer size, representing the number of elements in the linked list.

The second line contains arr space-separated integers, representing the elements of the linked list, and inserts nodes at the end.
Output Format
The output consists of the following:

The first line should display the elements of the original linked list.

The second line should display the elements of the linked list after deleting the second-to-last node.

Refer to the sample output for formatting specifications.

Constraints
1 <= size <= 100

-50000 <= arr <= 50000

Sample Input Sample Output

```
5
1 2 3 4 5

Original Linked List: 1 2 3 4 5
Updated Linked List: 1 2 3 5
```

Sample Input Sample Output

```
6
-50000 50000 7976 9887 4000 7765

Original Linked List: -50000 50000 7976 9887 4000 7765
Updated Linked List: -50000 50000 7976 9887 7765
```

Time Limit: - ms Memory Limit: - kb Code Size: - kb
Q10.
Problem Statement

Hema wants to create a program to delete nodes in a linked list that appear more than once. The program should take user input to construct a linked list, insert nodes at the end of the list, and then delete any duplicate nodes. Finally, it should display the updated linked list.

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

Input Format
The first line of input consists of the number of nodes in the linked list, numNodes (an integer).

The second line of input consists of the values for each node, inserted at the end of the list, separated by spaces, in the order they appear in the linked list.

Output Format
The output displays the original list and the linked list in the next line after removing the nodes that appear more than once.

Refer to the sample output for format specifications.

Constraints

1 <= numNodes <= 100

-10000 <= values of nodes <= 10000

Sample Input Sample Output

```
5
50 60 70 80 50

Original List: 50 60 70 80 50
Updated List: 50 60 70 80
```

Sample Input Sample Output

```
6
-10000 -10000 -10000 10000 10000 10000

Original List: -10000 -10000 -10000 10000 10000 10000
Updated List: -10000 10000
```

Sample Input Sample Output

```
3
4 5 6

Original List: 4 5 6
Updated List: 4 5 6
```

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

Q11.

Problem Statement

Sita wants to remove duplicate nodes from a sorted, singly linked list. Implement a program that takes a sorted linked list as input and removes any duplicate nodes, resulting in a modified linked list.

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

Input Format

The first line of input contains an integer representing the number of nodes in the linked list.

The second line contains a series of space-separated integers, representing the values of the nodes in the sorted linked list.

Output Format

The output displays the modified linked list after removing duplicate nodes.

Refer to the sample output for formatting specifications.

Constraints

The linked list should be sorted in ascending order.

1 <= number of nodes <= 100

-50000 <= values of nodes <= 50000

Sample Input Sample Output

```
5
1 2 2 3 4

Original Linked List: 1 2 2 3 4
Linked List after removing duplicates: 1 2 3 4
```

Sample Input Sample Output

```
3
-50000 50000 1000

Original Linked List: -50000 50000 1000
Linked List after removing duplicates: -50000 50000 1000
```

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

Q12.

Problem Statement

Dharun is working on a program to manipulate linked lists. He wants to write a function that takes two linked lists as input, inserts nodes at the end, and deletes all the nodes from the first list that also appear in the second list. Dharun needs your help to implement this function. The function should take two linked lists, **list1** and **list2,** as input, where each list is represented by its head node.

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

Input Format
The first line contains an integer n, denoting the number of nodes in list1.

The next line contains n space-separated integers, representing the values of the nodes in list1.

The next line contains an integer m, denoting the number of nodes in list2.

The next line contains m space-separated integers, representing the values of the nodes in list2.

Output Format
The first line of output displays the elements of the first linked list before the deletion, separated by a space.

The second line of output displays the elements of the first linked list after the deletion, separated by a space.

If all elements in the first linked list are the same after deletion, the third line will be displayed stating, "All elements in the first linked list are the same."

Refer to the sample output for formatting specifications.

Constraints
1 <= n,m <= 100

-50000 <= values of nodes <= 50000

Sample Input Sample Output

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

First Linked List before deletion: 2 3 4 5 1
First Linked List after deletion: 4 5
```

Sample Input Sample Output

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

First Linked List before deletion: 1 2 3 4 5
First Linked List after deletion:
All elements in the first linked list are the same.
```

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

Answer Key & Solution

Section 1 - Coding

Q1 Test Case Input Output

```
6
Welcome
to
the
World
of
Programming

Welcome to the World of
```

Weightage - 10 Input Output

```
2
Data
Structures

Data
```

Weightage - 10 Input Output

```
8
This
is
a
more
challenging
test
for
you

This is a more challenging test for
```

Weightage - 15 Input Output

```
7
Let's
see
how
well
you
can
handle

Let's see how well you can
```

Weightage - 15 Input Output

```
10
The
quick
brown
fox
jumps
over
the
lazy
dog
in

The quick brown fox jumps over the lazy dog
```

Weightage - 25 Input Output

```
15
I
```

```
am
a
sentence
with
multiple
words
and
characters
to
test
the
program
logic

I am a sentence with multiple words and characters to test the program logic
```

Weightage - 25 Sample Input Sample Output

```
4
Hello
World
I
Am

Hello World I
```

Sample Input Sample Output

```
3
Hello
Hello
Hello

Hello Hello
```

Solution

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

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

void append(Node** head, const string& data) {
    Node* tmp = new Node;
    tmp->data = data;
    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;
    cout << curr->data << " ";
    print(curr->next);
}

void delete_last(Node** head) {
    if (*head == nullptr || (*head)->next == nullptr) {
        delete *head;
        *head = nullptr;
    } else {
        Node* curr = *head;
        while (curr->next->next != nullptr) {
            curr = curr->next;
        }
        delete curr->next;
        curr->next = nullptr;
    }
}

int main() {
    int num_of_nodes;
    string data;
    cin >> num_of_nodes;
    Node* myList = nullptr;
    for (int i = 0; i < num_of_nodes; i++) {
        cin >> data;
        append(&myList, data);
    }
    delete_last(&myList);
    print(myList);
    return 0;
}
```

Q2 Test Case Input Output

```
150
477  1326  1213  1345  381  1090  61  1460  173  721  801  391  331  312  585  528  543  640  221  704  316
1045  1439  1428  919  1477  1231  508  1142  987  192  358  1469  571  1366  1070  998  1110  232  661  937
653  874  1413  1294  970  304  444  1355  139  939  924  1347  1023  1388  1387  589  1137  1287  419  680
21  742  518  1215  388  500  1091  153  98  755  951  1344  679  1488  963  240  401  127  538  561  852
1299  252  751  698  520  853  368  1342  426  1493  980  1189  547  778  972  339  1181  857  1417  1286
271  1039  1224  156  412  158  1265  314  1412  968  839  1059  552  1061  108  45  479  745  1048  814  220
1455  808  83  54  1125  619  815  369  186  1285  451  864  1479  414  1410  109  905  34  1253  445  928
542  1157  533  250  73  715
100

477 1326 1213 1345 381 1090 61 1460 173 721 801 391 331 312 585 528 543 640 221 704 316 1045 1439 1428 919
1477 1231 508 1142 987 192 358 1469 571 1366 1070 998 1110 232 661 937 653 874 1413 1294 970 304 444 1355 139
939 924 1347 1023 1388 1387 589 1137 1287 419 680 21 742 518 1215 388 500 1091 153 98 755 951 1344 679 1488
963 240 401 127 538 561 852 1299 252 751 698 520 853 368 1342 426 1493 980 1189 547 778 972 339 1181 1417
1286 271 1039 1224 156 412 158 1265 314 1412 968 839 1059 552 1061 108 45 479 745 1048 814 220 1455 808 83 54
1125 619 815 369 186 1285 451 864 1479 414 1410 109 905 34 1253 445 928 542 1157 533 250 73 715
```

Weightage - 25 Input Output

```
10
806  197  328  1280  1068  850  852  1404  1127  1451
1

197 328 1280 1068 850 852 1404 1127 1451
```

Weightage - 10 Input Output

```
15
27  67  9  98  1  15  50  83  34  58  26  75  35  71  5
10

27 67 9 98 1 15 50 83 34 26 75 35 71 5
```

Weightage - 15 Input Output

```
20
293  412  441  275  297  144  452  76  436  57  65  336  183  25  105  129  191  137  327  159
15

293 412 441 275 297 144 452 76 436 57 65 336 183 25 129 191 137 327 159
```

Weightage - 15 Input Output

```
100
5996  4826  4669  5371  6341  4916  2471  1755  3714  4922  5465  995  292  6183  2850  783  539  6979  847
5882  4266  1557  3517  5721  6196  3558  362  3758  5631  5810  2900  5553  5741  4021  3312  574  2429
2061  1608  1185  4560  6430  378  2031  6664  2382  6646  4576  3589  5050  5552  5273  854  4892  3070
2103  490  6511  5764  3084  2077  6090  1075  4143  314  415  3210  6151  3605  4257  2057  5951  4854  4405
2259  2211  828  5230  755  3928  344  3994  6733  5406  2481  2535  231  2433  3402  6705  2823  2513  4657
811  2913  5128  1072  2206  5619  5156
99

5996 4826 4669 5371 6341 4916 2471 1755 3714 4922 5465 995 292 6183 2850 783 539 6979 847 5882 4266 1557 3517
5721 6196 3558 362 3758 5631 5810 2900 5553 5741 4021 3312 574 2429 2061 1608 1185 4560 6430 378 2031 6664
2382 6646 4576 3589 5050 5552 5273 854 4892 3070 2103 490 6511 5764 3084 2077 6090 1075 4143 314 415 3210
6151 3605 4257 2057 5951 4854 4405 2259 2211 828 5230 755 3928 344 3994 6733 5406 2481 2535 231 2433 3402
6705 2823 2513 4657 811 2913 5128 1072 2206 5156
```

Weightage - 25 Input Output

```
7
10 20 30 40 50 60 70
4
```

```
10 20 30 50 60 70
```

Weightage - 10 Sample Input Sample Output

```
7
7270 5318 2964 4237 7388 7010 2468
4

7270 5318 2964 7388 7010 2468
```

Sample Input Sample Output

```
6
10 20 30 40 50 60
3

10 20 40 50 60
```

Solution

```
#include <iostream>

struct Node {
    int data;
    Node* next;

    Node(int val) : data(val), next(nullptr) {}
};

void append(Node** head_ref, Node** tail_ref, int new_data) {
    Node* new_node = new Node(new_data);
    if (*head_ref == nullptr)
        *head_ref = new_node;
    else
        (*tail_ref)->next = new_node;

    *tail_ref = new_node;
}

Node* deleteNode(Node* head, int x) {
    Node* temp1 = head;
    Node* temp2;
    if (x == 1) {
        head = head->next;
        delete temp1;
        return head;
    }

    for (int i = 0; i < (x - 1); i++) {
        temp2 = temp1;
        temp1 = temp1->next;
    }

    temp2->next = temp1->next;
    delete temp1;

    return head;
}

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

int main() {
    int n, l;
    Node* head = nullptr;
    Node* tail = nullptr;

    std::cin >> n;
    for (int i = 0; i < n; i++) {
        std::cin >> l;
        append(&head, &tail, l);
    }

    int kk;
    std::cin >> kk;

    head = deleteNode(head, kk);

    printList(head);
```

```
    return 0;
}
```

Q3 Test Case Input Output

```
6
One Two Three Four Five Six

Linked list data: One Two Three Four Five Six
After deleting alternate node:One Three Five
```

Weightage - 10 Input Output

```
4
A B C D

Linked list data: A B C D
After deleting alternate node:A C
```

Weightage - 10 Input Output

```
7
Monday Tuesday Wednesday Thursday Friday Saturday Sunday

Linked list data: Monday Tuesday Wednesday Thursday Friday Saturday Sunday
After deleting alternate node:Monday Wednesday Friday Sunday
```

Weightage - 15 Input Output

```
8
Alice Bob Charlie David Eve Frank George Henry

Linked list data: Alice Bob Charlie David Eve Frank George Henry
After deleting alternate node:Alice Charlie Eve George
```

Weightage - 15 Input Output

```
9
Red Orange Yellow Green Blue Indigo Violet Pink White

Linked list data: Red Orange Yellow Green Blue Indigo Violet Pink White
After deleting alternate node:Red Yellow Blue Violet White
```

Weightage - 25 Input Output

```
10
Apple Banana Cherry Date Elderberry Fig Grapefruit Honeydew Jackfruit Kiwi

Linked list data: Apple Banana Cherry Date Elderberry Fig Grapefruit Honeydew Jackfruit Kiwi
After deleting alternate node:Apple Cherry Elderberry Grapefruit Jackfruit
```

Weightage - 25 Sample Input Sample Output

```
2
Apple Banana

Linked list data: Apple Banana
After deleting alternate node:Apple
```

Sample Input Sample Output

```
5
Red Green Blue Yellow Orange

Linked list data: Red Green Blue Yellow Orange
After deleting alternate node:Red Blue Orange
```

Sample Input Sample Output

```
0

List is empty
```

Solution

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

struct node {
    string data;
    node* nextptr;
} *stnode; // node declared

void make(int n);
void alternateDel(node* stnode);
void display();

int main() // main method
{
    int n;
    cin >> n;

    make(n);

    if (stnode == nullptr) {
        cout << "List is empty";
    } else {
        cout << "Linked list data: ";
        display();
        cout << "\nAfter deleting alternate node:";
        alternateDel(stnode);
        display();
    }

    return 0;
}


void make(int n) // function to create linked list
{
    struct node* frntNode, * tmp;

    if (n == 0) {
        stnode = nullptr;
        return;
    }

    string data;

    cin >> data;

    stnode = new node;
    stnode->data = data;
    stnode->nextptr = nullptr;
    tmp = stnode;

    for (int i = 2; i <= n; i++) {

        cin >> data;

        frntNode = new node;
        frntNode->data = data;
        frntNode->nextptr = nullptr;
        tmp->nextptr = frntNode;
        tmp = tmp->nextptr;
    }
}

void display() // function to display linked list
{
    if (stnode == nullptr) {
```

```
        cout << "List is empty";
    } else {
        struct node* tmp = stnode;

        while (tmp != nullptr) {
            cout << tmp->data << " ";
            tmp = tmp->nextptr;
        }
    }
}

void alternateDel(node* stnode) // function to delete alternate nodes
{
    if (stnode == nullptr)
        return;

    node* prev = stnode;
    node* alt_node = stnode->nextptr;

    while (prev != nullptr && alt_node != nullptr) {
        prev->nextptr = alt_node->nextptr;

        delete alt_node;

        prev = prev->nextptr;
        if (prev != nullptr)
            alt_node = prev->nextptr;
    }
}
```

Q4 Test Case Input Output

```
5
9 88 53 6 87

88 53 6 87
```

Weightage - 10 Input Output

```
10
33  97  26  6  8  66  52  69  76  73

97 26 6 8 66 52 69 76 73
```

Weightage - 15 Input Output

```
10
499  255  456  323  347  470  475  306  200  181

255 456 323 347 470 475 306 200 181
```

Weightage - 15 Input Output

```
13
300  779  907  903  452  331  968  550  969  834  506  399  706

779 907 903 452 331 968 550 969 834 506 399 706
```

Weightage - 25 Input Output

```
15
4840  6072  310  4795  6768  4517  4614  58  5534  165  213  5758  6430  1600  2154

6072 310 4795 6768 4517 4614 58 5534 165 213 5758 6430 1600 2154
```

Weightage - 25 Input Output

```
8
95 100 110 120 130 140 150 160

100 110 120 130 140 150 160
```

Weightage - 10 Sample Input Sample Output

```
5
2 3 6 9 8

3 6 9 8
```

Sample Input Sample Output

```
20
92 8 14 44 17 55 47 93 22 5 11 34 45 87 80 16 63 15 31 10

8 14 44 17 55 47 93 22 5 11 34 45 87 80 16 63 15 31 10
```

Solution

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

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

void append(Node** head, int val) {
    Node* tmp = new 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;
    cout << curr->val << " ";
    print(curr->next);
}

void popLeft(Node** head) {
    if (*head == NULL) {
        return;
    }
    Node* tmp = *head;
    *head = (*head)->next;
    delete tmp;
}

int main(void) {
    int n, i, pos, new_val;
    cin >> n;
    Node* myList = NULL;
    for (i = 0; i < n; i++) {
        int val;
        cin >> val;
        append(&myList, val);
    }
    popLeft(&myList);
    print(myList);
    return 0;
}
```

Q5 Test Case Input Output

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

Linked List before deletion: 1 2 3 4 5 6 7
Linked List after deletion: 1 2 6 7
```

Weightage - 10 Input Output

```
6
2 4 6 8 10 12
1
6

Linked List before deletion: 2 4 6 8 10 12
Linked List after deletion: all the elements are deleted
```

Weightage - 10 Input Output

```
7
10 20 30 40 50 60 70
4
6

Linked List before deletion: 10 20 30 40 50 60 70
Linked List after deletion: 10 20 30 70
```

Weightage - 15 Input Output

```
100
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70
71 72 73 74 75 76 77 78 79 80
81 82 83 84 85 86 87 88 89 90
91 92 93 94 95 96 97 98 99 100
1
10

Linked List before deletion: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66
67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
Linked List after deletion: 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73
74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
```

Weightage - 25 Input Output

```
80
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70
71 72 73 74 75 76 77 78 79 80
81 82 83 84 85 86 87 88 89 90
91 92 93 94 95 96 97 98 99 100
25
45
```

```
Linked List before deletion: 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
Linked List after deletion: 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 66 67 68
69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
```

Weightage - 25 Input Output

```
60
41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70
71 72 73 74 75 76 77 78 79 80
81 82 83 84 85 86 87 88 89 90
91 92 93 94 95 96 97 98 99 100
20
40

Linked List before deletion: 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67
68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
Linked List after deletion: 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 81 82 83 84 85 86 87 88
89 90 91 92 93 94 95 96 97 98 99 100
```

Weightage - 15 Sample Input Sample Output

```
5
1 2 3 4 5
1
3

Linked List before deletion: 1 2 3 4 5
Linked List after deletion: 4 5
```

Sample Input Sample Output

```
5
-50000 50000 4000 3676 7263
1
5

Linked List before deletion: -50000 50000 4000 3676 7263
Linked List after deletion: all the elements are deleted
```

Solution

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

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

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

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

void deleteNodesInRange(Node** head, int start, int end) {
    if (*head == nullptr) {
        return;
    }

    Node* temp = *head;
    Node* prev = nullptr;

    for (int i = 1; i < start && temp != nullptr; i++) {
        prev = temp;
        temp = temp->next;
    }

    for (int i = start; i <= end && temp != nullptr; i++) {
        Node* nextNode = temp->next;
        delete temp;
        temp = nextNode;
    }

    if (prev != nullptr) {
        prev->next = temp;
    } else {
        *head = temp;
    }
}

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

void deleteLinkedList(Node* head) {
    Node* temp;
    while (head != nullptr) {
        temp = head;
        head = head->next;
        delete temp;
    }
```

```
}

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

    cin >> size;

    for (int i = 0; i < size; i++) {
        int value;
        cin >> value;
        insertNode(&head, value);
    }

    int start, end;
    cin >> start;
    cin >> end;

    cout << "Linked List before deletion: ";
    displayLinkedList(head);

    deleteNodesInRange(&head, start, end);

    if (head == nullptr) {
        cout << "Linked List after deletion: all the elements are deleted" << endl;
    } else {
        cout << "Linked List after deletion: ";
        displayLinkedList(head);
    }

    deleteLinkedList(head);

    return 0;
}
```

Q6 Test Case Input Output

```
5
7 6 5 4 8

Original Linked List: 7 6 5 4 8
Updated Linked List: 7 6 4 8
```

Weightage - 10 Input Output

```
5
-50000 50000 3747 3888 8788

Original Linked List: -50000 50000 3747 3888 8788
Updated Linked List: -50000 50000 3888 8788
```

Weightage - 10 Input Output

```
100
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

Original Linked List: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68
69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
Updated Linked List: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
```

Weightage - 25 Input Output

```
75
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75

Original Linked List: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68
69 70 71 72 73 74 75
Updated Linked List: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
33 34 35 36 37 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75
```

Weightage - 25 Input Output

```
41
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
40 41

Original Linked List: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
33 34 35 36 37 38 39 40 41
Updated Linked List: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 22 23 24 25 26 27 28 29 30 31 32 33
34 35 36 37 38 39 40 41
```

Weightage - 15 Input Output

```
35
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35

Original Linked List: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
33 34 35
Updated Linked List: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
34 35
```

Weightage - 15 Sample Input Sample Output

```
5
```

```
1 2 3 4 5

Original Linked List: 1 2 3 4 5
Updated Linked List: 1 2 4 5
```

Sample Input Sample Output

```
6
1 2 3 4 5 6

Original Linked List: 1 2 3 4 5 6
Updated Linked List: 1 2 3 5 6
```

Solution

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

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

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

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

void deleteMiddleNode(Node* head) {
    if (head == nullptr || head->next == nullptr) {
        return;
    }

    Node* slow = head;
    Node* fast = head;
    Node* prev = nullptr;

    while (fast != nullptr && fast->next != nullptr) {
        fast = fast->next->next;
        prev = slow;
        slow = slow->next;
    }

    prev->next = slow->next;
    delete slow;
}

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

int main() {
    Node* head = nullptr;
    int size;
    cin >> size;
    for (int i = 0; i < size; i++) {
        int value;
        cin >> value;
        insertNode(&head, value);
    }

    cout << "Original Linked List: ";
    displayList(head);

    deleteMiddleNode(head);
```

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

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

    return 0;
}
```

Q7 Test Case Input Output

```
7
1 2 3 4 5 6 7

Original Linked List: 1 2 3 4 5 6 7
Final Linked List: 1 3 5 7
```

Weightage - 10 Input Output

```
6
-5 -10 -15 -20 -25 -30

Original Linked List: -5 -10 -15 -20 -25 -30
Final Linked List: -5 -15 -25
```

Weightage - 10 Input Output

```
30
4 8 12 16 20 24 28 32 36 10
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20

Original Linked List: 4 8 12 16 20 24 28 32 36 10 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Final Linked List: 4 12 20 28 36 1 3 5 7 9 11 13 15 17 19
```

Weightage - 15 Input Output

```
80
4 8 12 16 20 24 28 32 36 10
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70

Original Linked List: 4 8 12 16 20 24 28 32 36 10 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69 70
Final Linked List: 4 12 20 28 36 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53
55 57 59 61 63 65 67 69
```

Weightage - 25 Input Output

```
100
4 8 12 16 20 24 28 32 36 10
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70
81 82 83 84 85 86 87 88 89 90
91 92 93 94 95 96 97 98 99 100

Original Linked List: 4 8 12 16 20 24 28 32 36 10 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69 70 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
Final Linked List: 4 12 20 28 36 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53
55 57 59 61 63 65 67 69 81 83 85 87 89 91 93 95 97 99
```

Weightage - 25 Input Output

```
60
```

```
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70
81 82 83 84 85 86 87 88 89 90
91 92 93 94 95 96 97 98 99 100

Original Linked List: 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69 70 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
Final Linked List: 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 81 83 85 87 89 91 93 95 97 99
```

Weightage - 15 Sample Input Sample Output

```
5
100 200 300 10000 9999

Original Linked List: 100 200 300 10000 9999
Final Linked List: 100 300 9999
```

Sample Input Sample Output

```
3
-10000 -2000 -3000

Original Linked List: -10000 -2000 -3000
Final Linked List: -10000 -3000
```

Solution

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

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

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

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

void deleteEvenPositionNodes(Node** head) {
    if (*head == nullptr || (*head)->next == nullptr) {
        return;
    }

    Node* prev = *head;
    Node* curr = (*head)->next;
    int position = 2;

    while (curr != nullptr) {
        if (position % 2 == 0) {
            prev->next = curr->next;
            delete curr;
            curr = prev->next;
        } else {
            prev = curr;
            curr = curr->next;
        }
        position++;
    }
}
void displayLinkedList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

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

    cin >> size;

    for (int i = 0; i < size; i++) {
        int value;
        cin >> value;
        insertNode(&head, value);
    }
```

```
    cout << "Original Linked List: ";
    displayLinkedList(head);

    deleteEvenPositionNodes(&head);

    cout << "Final Linked List: ";
    displayLinkedList(head);

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

    return 0;
}
```

Q8 Test Case Input Output

```
6
5 7 8 -10 -23 1
9

Original Linked List: 5 7 8 -10 -23 1
Modified Linked List: 5 7 8 -10 -23 1
```

Weightage - 10 Input Output

```
6
7 14 21 28 35 42
30

Original Linked List: 7 14 21 28 35 42
Modified Linked List: 7 14 21 28
```

Weightage - 10 Input Output

```
10
2 4 6 8 10 12 14 16 18 20
20

Original Linked List: 2 4 6 8 10 12 14 16 18 20
Modified Linked List: 2 4 6 8 10 12 14 16 18 20
```

Weightage - 15 Input Output

```
50
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
25

Original Linked List: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
Modified Linked List: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
```

Weightage - 15 Input Output

```
100
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70
71 72 73 74 75 76 77 78 79 80
81 82 83 84 85 86 87 88 89 90
91 92 93 94 95 96 97 98 99 100
10

Original Linked List: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68
69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
Modified Linked List: 1 2 3 4 5 6 7 8 9 10
```

Weightage - 25 Input Output

```
80
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
```

```
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70
71 72 73 74 75 76 77 78 79 80
48

Original Linked List: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68
69 70 71 72 73 74 75 76 77 78 79 80
Modified Linked List: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
```

Weightage - 25 Sample Input Sample Output

```
5
8 7 5 3 2
5

Original Linked List: 8 7 5 3 2
Modified Linked List: 5 3 2
```

Sample Input Sample Output

```
5
-8 7 -5 -3 -2
0

Original Linked List: -8 7 -5 -3 -2
Modified Linked List: -8 -5 -3 -2
```

Solution

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;
};

Node* getNode(int data) {
    Node* newNode = new Node;
    newNode->data = data;
    newNode->next = nullptr;
    return newNode;
}

void deleteGreaterNodes(Node** head_ref, int x) {
    Node* temp = *head_ref;
    Node* prev = nullptr;


    while (temp != nullptr && temp->data > x) {
        *head_ref = temp->next;
        delete temp;
        temp = *head_ref;
    }


    while (temp != nullptr) {

        while (temp != nullptr && temp->data <= x) {
            prev = temp;
            temp = temp->next;
        }


        if (temp == nullptr)
            return;


        prev->next = temp->next;
        delete temp;

        temp = prev->next;
    }
}

void printList(Node* head) {
    Node* temp = head;
    while (temp) {
        cout << temp->data << " ";
        temp = temp->next;
    }
}

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

    cin >> size;

    for (int i = 0; i < size; i++) {
        int value;
        cin >> value;
        if (head == nullptr) {
            head = getNode(value);
        } else {
            Node* temp = head;
```

```
            while (temp->next != nullptr) {
                temp = temp->next;
            }
            temp->next = getNode(value);
        }
    }

    int x;
    cin >> x;

    cout << "Original Linked List: ";
    printList(head);

    deleteGreaterNodes(&head, x);

    cout << "\nModified Linked List: ";
    printList(head);

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

    return 0;
}
```

Q9 Test Case Input Output

```
10
3 5 6 8 5 9 10 2 4 1

Original Linked List: 3 5 6 8 5 9 10 2 4 1
Updated Linked List: 3 5 6 8 5 9 10 2 1
```

Weightage - 15 Input Output

```
6
1 4 5 7 3 2

Original Linked List: 1 4 5 7 3 2
Updated Linked List: 1 4 5 7 2
```

Weightage - 10 Input Output

```
5
6 7 4 3 2

Original Linked List: 6 7 4 3 2
Updated Linked List: 6 7 4 2
```

Weightage - 10 Input Output

```
20
8 7 6 5 9 1 5 7 3 4 6 4 12 65 87 65 87 53 32 11

Original Linked List: 8 7 6 5 9 1 5 7 3 4 6 4 12 65 87 65 87 53 32 11
Updated Linked List: 8 7 6 5 9 1 5 7 3 4 6 4 12 65 87 65 87 53 11
```

Weightage - 15 Input Output

```
100
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70
71 72 73 74 75 76 77 78 79 80
81 82 83 84 85 86 87 88 89 90
91 92 93 94 95 96 97 98 99 100

Original Linked List: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68
69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
Updated Linked List: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68
69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 100
```

Weightage - 25 Input Output

```
70
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70
```

```
Original Linked List: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68
69 70
Updated Linked List: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68
70
```

Weightage - 25 Sample Input Sample Output

```
5
1 2 3 4 5

Original Linked List: 1 2 3 4 5
Updated Linked List: 1 2 3 5
```

Sample Input Sample Output

```
6
-50000 50000 7976 9887 4000 7765

Original Linked List: -50000 50000 7976 9887 4000 7765
Updated Linked List: -50000 50000 7976 9887 7765
```

Solution

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

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

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

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

void deleteSecondLastNode(Node** head) {
    if (*head == nullptr || (*head)->next == nullptr) {
        return;
    }

    Node* prev = nullptr;
    Node* curr = *head;

    while (curr->next->next != nullptr) {
        prev = curr;
        curr = curr->next;
    }

    prev->next = curr->next;
    delete curr;
}

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

void deleteLinkedList(Node* head) {
    Node* temp;
    while (head != nullptr) {
        temp = head;
        head = head->next;
        delete temp;
    }
}

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

    cin >> size;

    for (int i = 0; i < size; i++) {
```

```
        int value;
        cin >> value;
        insertNode(&head, value);
    }

    cout << "Original Linked List: ";
    displayLinkedList(head);

    deleteSecondLastNode(&head);

    cout << "Updated Linked List: ";
    displayLinkedList(head);

    deleteLinkedList(head);

    return 0;
}
```

Q10 Test Case Input Output

```
5
30 35 40 45 50

Original List: 30 35 40 45 50
Updated List: 30 35 40 45 50
```

Weightage - 10 Input Output

```
9
-30 -35 -40 -35 -50 -65 -60 -65 -70

Original List: -30 -35 -40 -35 -50 -65 -60 -65 -70
Updated List: -30 -35 -40 -50 -65 -60 -70
```

Weightage - 10 Input Output

```
100
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

Original List: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70
71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
Updated List: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
```

Weightage - 25 Input Output

```
80
10 20 30 40 50 60 70 80 90 100 10 20 30 40 50 60 70 80 90 100 10 20 30 40 50 60 70 80 90 100 10 20 30 40 50
60 70 80 90 100 10 20 30 40 50 60 70 80 90 100 10 20 30 40 50 60 70 80 90 100 10 20 30 40 50 60 70 80 90 100
10 20 30 40 50 60 70 80 90 100

Original List: 10 20 30 40 50 60 70 80 90 100 10 20 30 40 50 60 70 80 90 100 10 20 30 40 50 60 70 80 90 100
10 20 30 40 50 60 70 80 90 100 10 20 30 40 50 60 70 80 90 100 10 20 30 40 50 60 70 80 90 100 10 20 30 40 50
60 70 80 90 100 10 20 30 40 50 60 70 80 90 100
Updated List: 10 20 30 40 50 60 70 80 90 100
```

Weightage - 25 Input Output

```
25
1 2 3 4 5 1 2 3 4 5 11 21 13 41 15 11 12 13 14 15 8 5 37 39 3

Original List: 1 2 3 4 5 1 2 3 4 5 11 21 13 41 15 11 12 13 14 15 8 5 37 39 3
Updated List: 1 2 3 4 5 11 21 13 41 15 12 14 8 37 39
```

Weightage - 15 Input Output

```
20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Original List: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Updated List: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

Weightage - 15 Sample Input Sample Output

```
5
50 60 70 80 50

Original List: 50 60 70 80 50
Updated List: 50 60 70 80
```

Sample Input Sample Output

```
6
-10000 -10000 -10000 10000 10000 10000

Original List: -10000 -10000 -10000 10000 10000 10000
Updated List: -10000 10000
```

Sample Input Sample Output

```
3
4 5 6

Original List: 4 5 6
Updated List: 4 5 6
```

Solution

```cpp
#include <iostream>

using namespace std;


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

void insertNode(Node** head, int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;

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

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

    Node* currNode = *head;

    while (currNode != nullptr) {
        Node* temp = currNode;
        while (temp->next != nullptr) {
            if (temp->next->data == currNode->data) {
                Node* duplicateNode = temp->next;
                temp->next = temp->next->next;
                delete duplicateNode;
            } else {
                temp = temp->next;
            }
        }
        currNode = currNode->next;
    }
}

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

int main() {
    Node* head = nullptr;
    int numNodes, value;

    cin >> numNodes;
    for (int i = 0; i < numNodes; i++) {
        cin >> value;
        insertNode(&head, value);
    }
```

```
    cout << "Original List: ";
    displayList(head);

    deleteDuplicates(&head);

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

    return 0;
}
```

Q11 Test Case Input Output

```
8
1 2 2 3 3 3 4 5

Original Linked List: 1 2 2 3 3 3 4 5
Linked List after removing duplicates: 1 2 3 4 5
```

Weightage - 100 Sample Input Sample Output

```
5
1 2 2 3 4

Original Linked List: 1 2 2 3 4
Linked List after removing duplicates: 1 2 3 4
```

Sample Input Sample Output

```
3
-50000 50000 1000

Original Linked List: -50000 50000 1000
Linked List after removing duplicates: -50000 50000 1000
```

Solution

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

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

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

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

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

    Node* current = head;

    while (current->next != nullptr) {
        if (current->data == current->next->data) {
            Node* duplicateNode = current->next;
            current->next = current->next->next;
            delete duplicateNode;
        } else {
            current = current->next;
        }
    }
}

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

void deleteLinkedList(Node* head) {
    Node* temp;
    while (head != nullptr) {
        temp = head;
        head = head->next;
        delete temp;
    }
}

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

    cin >> size;
```

```
    for (int i = 0; i < size; i++) {
        int value;
        cin >> value;
        insertNode(&head, value);
    }

    cout << "Original Linked List: ";
    displayLinkedList(head);

    removeDuplicates(head);

    cout << "Linked List after removing duplicates: ";
    displayLinkedList(head);

    deleteLinkedList(head);

    return 0;
}
```

Q12 Test Case Input Output

```
3
5 4 2
2
4 1

First Linked List before deletion: 5 4 2
First Linked List after deletion: 5 2
```

Weightage - 10 Input Output

```
3
-50000 50000 3000
2
-50000 50000

First Linked List before deletion: -50000 50000 3000
First Linked List after deletion: 3000
All elements in the first linked list are the same.
```

Weightage - 10 Input Output

```
100
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
100
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

First Linked List before deletion: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64
65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
First Linked List after deletion:
All elements in the first linked list are the same.
```

Weightage - 25 Input Output

```
100
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

```
First Linked List before deletion: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64
65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
First Linked List after deletion: 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81
82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
```

Weightage - 25 Input Output

```
40
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
40
60
41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76
77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

First Linked List before deletion: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31 32 33 34 35 36 37 38 39 40
First Linked List after deletion: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31 32 33 34 35 36 37 38 39 40
```

Weightage - 15 Input Output

```
30
41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70
60
41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76
77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

First Linked List before deletion: 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65
66 67 68 69 70
First Linked List after deletion:
All elements in the first linked list are the same.
```

Weightage - 15 Sample Input Sample Output

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

First Linked List before deletion: 2 3 4 5 1
First Linked List after deletion: 4 5
```

Sample Input Sample Output

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

First Linked List before deletion: 1 2 3 4 5
First Linked List after deletion:
All elements in the first linked list are the same.
```

Solution

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

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

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

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

bool valueExists(Node* second, int value) {
    Node* curr = second;
    while (curr != nullptr) {
        if (curr->data == value) {
            return true;
        }
        curr = curr->next;
    }
    return false;
}

void deleteNodesInSecondList(Node** first, Node* second) {
    if (*first == nullptr || second == nullptr) {
        return;
    }

    Node* prev = nullptr;
    Node* curr1 = *first;

    while (curr1 != nullptr) {
        if (valueExists(second, curr1->data)) {
            if (prev == nullptr) {
                *first = curr1->next;
                delete curr1;
                curr1 = *first;
            } else {
                prev->next = curr1->next;
                delete curr1;
                curr1 = prev->next;
            }
        } else {
            prev = curr1;
            curr1 = curr1->next;
        }
    }
}

void displayLinkedList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
```

```
    }
    cout << endl;
}

void deleteLinkedList(Node* head) {
    Node* temp;
    while (head != nullptr) {
        temp = head;
        head = head->next;
        delete temp;
    }
}

bool areAllElementsSame(Node* head) {
    if (head == nullptr) {
        return true;
    }

    int firstElement = head->data;
    Node* current = head->next;
    while (current != nullptr) {
        if (current->data != firstElement) {
            return false;
        }
        current = current->next;
    }
    return true;
}

int main() {
    Node* first = nullptr;
    Node* second = nullptr;
    int size1, size2;

    cin >> size1;

    for (int i = 0; i < size1; i++) {
        int value;
        cin >> value;
        insertNode(&first, value);
    }

    cin >> size2;

    for (int i = 0; i < size2; i++) {
        int value;
        cin >> value;
        insertNode(&second, value);
    }

    cout << "First Linked List before deletion: ";
    displayLinkedList(first);

    deleteNodesInSecondList(&first, second);

    cout << "First Linked List after deletion: ";
    displayLinkedList(first);

    if (areAllElementsSame(first)) {
        cout << "All elements in the first linked list are the same.";
    }

    deleteLinkedList(first);
    deleteLinkedList(second);
```

```
    return 0;
}
```