Lab Assingment-4

ROLL: 2005535 | NAME: SAHIL SINGH | DATE: 17/08/21

QUES 1: [1] Write a menu driven program to perform the following operations in a single linked list by using suitable user defined functions for each case.

- Traverse the list.
- Check if the list is empty. [This function will be called to check underflow condition during the delete operation]
- Insert a node after a given data item
- Insert a node before a given data item
- Delete a node after a given data item
- Delete a node before a given data item
- Insert a node at the certain position (at beginning/end/any position).
- Delete a node at the certain position (at beginning/end/any position).
- Delete a node for the given key.
- Search for an element in the linked list.
- Sort the elements of the linked list
- Print the elements of the linked list in the reverse order.
- Reverse the nodes of the linked list.
- Print n th node from the last of a linked list.
- Delete the duplicate elements in a linked list.

SOLUTION:

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct Node
{
    int data;
    struct Node *link;
} Node;
void push(Node **, int);
void display(Node *);
int isEmpty(Node *);

//Extra

void push(Node **start, int n)
{
    /*Making the new node*/
    Node *temp = (Node *)malloc(sizeof(Node));
    temp->data = n;
    temp->link = NULL;
    /*Connecting the new node*/
    if (!*start)
    {
        *start = temp;
        return;
    }
    Node *tempStart = *start;
    /*Travel to last node*/
    while (tempStart->link)
        tempStart = tempStart->link;
    tempStart->link = temp; //Insert after last node
}
void display(Node *start)
{
    while (!isEmpty(start))
    {
        printf("%d->", start->data);
        start = start->link;
    }
    printf("null\n");
}
int isEmpty(Node *start)
{
    if (!start)
        return 1; //If the first node pointer is null return true, false otherwise.
    return 0;
}
void push_after(Node **start, int pos, int val)
{
    /*Making the new node*/
    Node *temp = (Node *)malloc(sizeof(Node));
    temp->data = val;
    temp->link = NULL;
    /*Connecting the new node*/
    if (!*start)
    {
        printf("List is empty!\n");
        return;
    }
    Node *tempStart = *start;
    while (tempStart && tempStart->data != pos)
    { //While end is not reached or pos is not found
        tempStart = tempStart -> link;
    }

    if (!tempStart)
    {

        //If end is reached, exit;
        printf("The element %d is not found!\n", pos);
        return;
    }
    /*Insert the new node after the tempStart node*/
    temp->link = tempStart->link;
    tempStart->link = temp;
}
void push_before(Node **start, int pos, int val)
```

```c
{
    /*Making the new node*/
    Node *temp = (Node *)malloc(sizeof(Node));
    temp->data = val;
    temp->link = NULL;
    /*Connecting the new node*/
    if (!*start)
    {
        printf("List is empty!\n");
        return;
    }
    Node *tempNext = *start;
    Node *tempPrev = *start;
    /*While end is not reached or pos is not found*/
    while (tempNext && tempNext->data != pos)
    {
        tempPrev = tempNext;
        tempNext = tempNext->link;
    }
    if (!tempNext)
    {

        //If end is reached, exit;
        printf("The element %d is not found!\n", pos);
        return;
    }
    else if (tempNext == *start) //If the first node has pos
    {
        temp->link = *start;
        *start = temp;
        return;
    }
    /*Insert the new node after the tempPrev node*/
    temp->link = tempPrev->link;
    tempPrev->link = temp;
}
void pop_after(Node **start, int pos)
{
    if (!*start)
    {
        printf("List is empty!\n");
        return;
    }
    Node *tempStart = *start;
    while (tempStart && tempStart->data != pos) //Travel to the value position
    {
        tempStart = tempStart->link;
    }
    if (!tempStart || !tempStart->link)
    { //If the list ends or the value position’s next is null
```

```c
        printf("Element %d is not found or it does'nt have an after node!\n", pos);
        return;
    }
    /*Update the value node to not include the after node.*/
    Node *ptr = tempStart->link;
    tempStart->link = tempStart->link->link;
    free(ptr);
}
void pop_before(Node **start, int val)
{
    if (!*start)
    {
        printf("List is empty!\n");
        return;
    }
    Node *tempNext = *start;
    Node *tempPrev = *start;
    /*While next node not null and next node data not value*/
    while (tempNext->link && tempNext->link->data != val)
    {
        tempPrev = tempNext;
        tempNext = tempNext->link;
    }
    if (!tempNext->link)
    {
        printf("The element %d is not found or it does'nt have a before node!\n", val);
        return;
    }
    else if (tempNext == tempPrev) //Update start if the before node is the first node
    {
        *start = tempNext->link;
    }

    else
        tempPrev->link = tempNext->link;

    //Update previous node link of before node

    free(tempNext);
}
void insert_at(Node **start, int pos, int val)
{
    /*Making the new node*/
    Node *temp = (Node *)malloc(sizeof(Node));
    temp->data = val;
    temp->link = NULL;
    /*Connecting the new node*/
    if (pos == 0 || !*start)
    { //For any value of pos, val will be first node, given the list is empty
        temp->link = *start;
        *start = temp;
        return;
    }
    else if (pos <= -1)
    {
        //Insert the node at the end for any value less than 0

        push(start, val);
        return;
    }
    Node *tempStart = *start;
    while (pos > 1 && tempStart->link)
    {
        //Travel to the list index

        tempStart = tempStart->link;
        pos--;
    }
    temp->link = tempStart->link;
    tempStart->link = temp;

    //Insert at the index
}
void pop_at(Node **start, int pos)
{
    if (!*start)
    {
        printf("The list is empty!\n");
        return;
    }
    struct Node *temp = *start;
    if (pos == 0)
    {
        //If the to be deleted element is the first node.
        *start = temp->link;
        free(temp);
        return;
    }
    int mode = pos;
    struct Node *ptr;
    while ((pos > 1 || mode == -1) && temp->link)
    {
        ptr = temp;
        temp = temp->link;
        pos--;
    }
    if (mode == -1)
    {
        ptr->link = temp->link;
        if (temp == *start)
```

```c
            *start = NULL; //If this is the only element in list
        free(temp);
        return;
    }
    else if (!temp->link || pos < -1)
    { //If the entered position is not found in the list, exit;
        printf("Invalid position !\n ");
        return;
    }
    ptr = temp->link;
    temp->link = ptr->link;
    free(ptr);
}
void pop_key(Node **start, int val)
{
    if (!*start)
    {
        printf("The list is empty!\n");
        return;
    }
    else if ((*start)->data == val)
    {

        //If the first element matches the key

        Node *ptr = *start;
        *start = (*start)->link;
        free(ptr);
        return;
    }
    Node *tempStart = *start;
    Node *tempPrev = *start;
    while (tempStart && tempStart->data != val)
    {
        tempPrev = tempStart;
        tempStart = tempStart->link;
    }
    if (!tempStart)
    {

        //If the whole list is traversed without key encounter, exit;

        printf("Key not found!\n");
        return;
    }
    /*Provide keys link to the previous node then free(key);*/
    tempPrev->link = tempStart->link;
    free(tempStart);
}
Node *search(Node *start, int val)
```

```c
{
    if (!start)
        return NULL;
    Node *list = NULL;
    int pos = 0;

    //Create a new list

    while (start)
    {
        //Insert the indices of the matched item in the new list
        if (start->data == val)
            push(&list, pos);
        start = start->link;
        pos++;
    }
    return list; //Return the new list with indices
}
void empty_the_list(Node **start)
{
    while (*start)
    {
        Node *ptr = *start;
        *start = (*start)->link;
        free(ptr);
    }
}
void selection_sort(Node **start)
{
    if (!*start)
        return;
    Node *ptr1 = *start;
    while (ptr1)
    {
        Node *ptr2 = ptr1->link;
        while (ptr2)
        {
            if (ptr1->data > ptr2->data)
            {
                int temp = ptr1->data;
                ptr1->data = ptr2->data;
                ptr2->data = temp;
            }
            ptr2 = ptr2->link;
        }
        ptr1 = ptr1->link;
    }
}
void print_in_rev(Node *start)
{
    if (!start)
    {
        //Go to the end of the list. Print value as stack collapses.
        printf("null<-");
        return;
    }
    print_in_rev(start->link);
    printf("%d<-", start->data);
}
void reverse_the_nodes(Node **start)
{
    if (!*start)
        return;
    Node *temp = *start;
    Node *temp1 = (*start)->link;
    Node *temp2;

    //current
    //next
    //previous

    while (temp1 != NULL)
    {
        temp2 = temp1->link;
        temp1->link = temp;
        temp = temp1;
        temp1 = temp2;
    }
    (*start)->link = NULL;
    *start = temp;
}
void nth_node_from_last(Node *start, int n)
{
    if (!start)
        return;
    int count = 0;
    Node *tempStart = start;
    while (start)
    {

        //Finding length of the list

        count++;
        start = start->link;
    }
    if (count - n < 0 || n <= 0)
    {
        printf("The node is not possible!\n");
        return;
    }
    while (count - n)
    {

        //Searching for the nth node from last

        count--;

        tempStart = tempStart->link;
    }
    printf("%d\n", tempStart->data);
}
void make_unique(Node **start)
{
    Node *tempStart =
        *start;
    int count_TS = 0; //Considering start node as 0th index
    while (tempStart)
    {
        Node *nextPtr = tempStart->link;
        int count_NP = count_TS + 1;
        while (nextPtr)
        {
            if (tempStart->data == nextPtr->data)
            {
                if (!nextPtr->link) //If no more elements present after deletion.
                    nextPtr = NULL;
                //Passes the index of the element to delete.
                pop_at(start, count_NP);
            }
            if (!nextPtr)
                break; //End the loop if no more elements present after deletion
            nextPtr = nextPtr->link;
            count_NP++;
        }
        tempStart = tempStart->link;
        count_TS++;
    }
}
int main()
{
    Node *list1 = NULL;
    int choice, pos, val;
    do
    {
        printf("Enter your choice:\n1) Create a new node\n2) Display the list\n");
        printf("3) Is the list empty?\n");
        printf("4) Insert the node after...\n5) Insert the node before...\n6) Delete node 
after...\n");
        printf("7) Delete node before...\n8) Insert at position\n9) Delete at position\n");
        printf("10) Delete a node for the given key\n11) Search for...\n12) Sort the
list\n");
        printf("13) Print the elements in reverse\n14) Reverse the list\n");
        printf("15) Print nth node from end\n16) Delete all duplicate items\n");
        printf("17) Drop the list\n18) Exit\n->: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("\nEnter an element: ");
            scanf("%d", &val);
            push(&list1, val);
            break;
        case 2:
            printf("\n");
            display(list1);
            break;
        case 3:
            printf("\nEmpty: ");
            printf((isEmpty(list1) ? "true\n" : "false\n"));
            break;
        case 4:
            printf("\nEnter value after which to insert: ");
            scanf("%d", &pos);
            printf("Enter the element to insert: ");
            scanf("%d", &val);
            push_after(&list1, pos, val);
            break;
        case 5:
            printf("\nEnter value before which to insert: ");
            scanf("%d", &pos);
            printf("Enter the element to insert: ");
            scanf("%d", &val);
            push_before(&list1, pos, val);
            break;
        case 6:
            printf("\nEnter value after which to delete: ");
            scanf("%d", &pos);
            pop_after(&list1, pos);
            break;
        case 7:
            printf("\nEnter value before which to delete: ");
            scanf("%d", &val);
            pop_before(&list1, val);
            break;
        case 8:
            printf("\nEnter the position at which to insert (-1 to insert at end): ");
            scanf("%d", &pos);
            printf("Enter value to insert: ");
            scanf("%d", &val);
```

```c
            insert_at(&list1, pos, val);
            break;
        case 9:
            printf("Enter the position to delete at (-1 to delete from end): ");
            scanf("%d", &pos);
            pop_at(&list1, pos);

            break;
        case 10:
            printf("\nEnter the key element: ");
            scanf("%d", &val);
            pop_key(&list1, val);
            break;
        case 11:
            printf("\nSearch for element: ");
            scanf("%d", &val);
            printf("Element found at position(s): ");
            Node *locList = search(list1, val);
            display(locList);
            empty_the_list(&locList);
            break;
        case 12:
            printf("\nThe list was sorted!\n");
            selection_sort(&list1);
            break;
        case 13:
            printf("\n");
            print_in_rev(list1);
            printf("\b\b ");
            printf("\n");
            break;
        case 14:
            printf("\nThe list was reversed!\n");
            reverse_the_nodes(&list1);
            break;
        case 15:
            printf("\nEnter node number: ");
            scanf("%d", &pos);
            printf("The node no. %d from the end is: ", pos);
            nth_node_from_last(list1, pos);
            break;
        case 16:
            printf("\nAll list elements are now unique!\n");
            make_unique(&list1);
            break;
        case 17:
            printf("\nList cleared!\n");
            empty_the_list(&list1);
            break;
        default:
            printf("\nExiting...\n");
        }
        printf("---------------------------\n");
    } while (choice >= 1 && choice <= 17);
    return 0;
}
```

OUTPUT:

```
Enter your choice:
1) Create a new node
2) Display the list
3) Is the list empty?
4) Insert the node after...
5) Insert the node before...
6) Delete node after...
7) Delete node before...
8) Insert at position
9) Delete at position
10) Delete a node for the given key
11) Search for...
12) Sort the list
13) Print the elements in reverse
14) Reverse the list
15) Print nth node from end
16) Delete all duplicate items
17) Drop the list
18) Exit
->: 1

Enter an element: 10
---------------------------
Enter your choice:
1) Create a new node
2) Display the list
3) Is the list empty?
4) Insert the node after...
5) Insert the node before...
6) Delete node after...
7) Delete node before...
8) Insert at position
9) Delete at position
10) Delete a node for the given key
11) Search for...
12) Sort the list
13) Print the elements in reverse
14) Reverse the list
15) Print nth node from end
16) Delete all duplicate items
```

```
17) Drop the list
18) Exit
->: 1

Enter an element: 20
---------------------------
Enter your choice:
1) Create a new node
2) Display the list
3) Is the list empty?
4) Insert the node after...
5) Insert the node before...
6) Delete node after...
7) Delete node before...
8) Insert at position
9) Delete at position
10) Delete a node for the given key
11) Search for...
12) Sort the list
13) Print the elements in reverse
14) Reverse the list
15) Print nth node from end
16) Delete all duplicate items
17) Drop the list
18) Exit
->: 1

Enter an element: 30
---------------------------
Enter your choice:
1) Create a new node
2) Display the list
3) Is the list empty?
4) Insert the node after...
5) Insert the node before...
6) Delete node after...
7) Delete node before...
8) Insert at position
9) Delete at position
10) Delete a node for the given key
11) Search for...
12) Sort the list
13) Print the elements in reverse
14) Reverse the list
15) Print nth node from end
16) Delete all duplicate items
17) Drop the list
18) Exit
->: 2
```

```
10->20->30->null
---------------------------
Enter your choice:
1) Create a new node
2) Display the list
3) Is the list empty?
4) Insert the node after...
5) Insert the node before...
6) Delete node after...
7) Delete node before...
8) Insert at position
9) Delete at position
10) Delete a node for the given key
11) Search for...
12) Sort the list
13) Print the elements in reverse
14) Reverse the list
15) Print nth node from end
16) Delete all duplicate items
17) Drop the list
18) Exit
->: 3

Empty: false
---------------------------
Enter your choice:
1) Create a new node
2) Display the list
3) Is the list empty?
4) Insert the node after...
5) Insert the node before...
6) Delete node after...
7) Delete node before...
8) Insert at position
9) Delete at position
10) Delete a node for the given key
11) Search for...
12) Sort the list
13) Print the elements in reverse
14) Reverse the list
15) Print nth node from end
16) Delete all duplicate items
17) Drop the list
18) Exit
->: 4

Enter value after which to insert: 10
Enter the element to insert: 15
---------------------------
Enter your choice:
```

```
1) Create a new node
2) Display the list
3) Is the list empty?
4) Insert the node after...
5) Insert the node before...
6) Delete node after...
7) Delete node before...
8) Insert at position
9) Delete at position
10) Delete a node for the given key
11) Search for...
12) Sort the list
13) Print the elements in reverse
14) Reverse the list
15) Print nth node from end
16) Delete all duplicate items
17) Drop the list
18) Exit
->: 2

10->15->20->30->null
---------------------------
Enter your choice:
1) Create a new node
2) Display the list
3) Is the list empty?
4) Insert the node after...
5) Insert the node before...
6) Delete node after...
7) Delete node before...
8) Insert at position
9) Delete at position
10) Delete a node for the given key
11) Search for...
12) Sort the list
13) Print the elements in reverse
14) Reverse the list
15) Print nth node from end
16) Delete all duplicate items
17) Drop the list
18) Exit
->: 5

Enter value before which to insert: 30
Enter the element to insert: 25
---------------------------
Enter your choice:
1) Create a new node
2) Display the list
3) Is the list empty?
```

```
4) Insert the node after...
5) Insert the node before...
6) Delete node after...
7) Delete node before...
8) Insert at position
9) Delete at position
10) Delete a node for the given key
11) Search for...
12) Sort the list
13) Print the elements in reverse
14) Reverse the list
15) Print nth node from end
16) Delete all duplicate items
17) Drop the list
18) Exit
->: 2

10->15->20->25->30->null
---------------------------
Enter your choice:
1) Create a new node
2) Display the list
3) Is the list empty?
4) Insert the node after...
5) Insert the node before...
6) Delete node after...
7) Delete node before...
8) Insert at position
9) Delete at position
10) Delete a node for the given key
11) Search for...
12) Sort the list
13) Print the elements in reverse
14) Reverse the list
15) Print nth node from end
16) Delete all duplicate items
17) Drop the list
18) Exit
->: 6

Enter value after which to delete: 10
---------------------------
Enter your choice:
1) Create a new node
2) Display the list
3) Is the list empty?
4) Insert the node after...
5) Insert the node before...
6) Delete node after...
7) Delete node before...
```

```
8) Insert at position
9) Delete at position
10) Delete a node for the given key
11) Search for...
12) Sort the list
13) Print the elements in reverse
14) Reverse the list
15) Print nth node from end
16) Delete all duplicate items
17) Drop the list
18) Exit
->: 2

10->20->25->30->null
---------------------------
Enter your choice:
1) Create a new node
2) Display the list
3) Is the list empty?
4) Insert the node after...
5) Insert the node before...
6) Delete node after...
7) Delete node before...
8) Insert at position
9) Delete at position
10) Delete a node for the given key
11) Search for...
12) Sort the list
13) Print the elements in reverse
14) Reverse the list
15) Print nth node from end
16) Delete all duplicate items
17) Drop the list
18) Exit
->: 7

Enter value before which to delete: 30
---------------------------
Enter your choice:
1) Create a new node
2) Display the list
3) Is the list empty?
4) Insert the node after...
5) Insert the node before...
6) Delete node after...
7) Delete node before...
8) Insert at position
9) Delete at position
10) Delete a node for the given key
11) Search for...
```

```
12) Sort the list
13) Print the elements in reverse
14) Reverse the list
15) Print nth node from end
16) Delete all duplicate items
17) Drop the list
18) Exit
->: 2

10->20->30->null
---------------------------
Enter your choice:
1) Create a new node
2) Display the list
3) Is the list empty?
4) Insert the node after...
5) Insert the node before...
6) Delete node after...
7) Delete node before...
8) Insert at position
9) Delete at position
10) Delete a node for the given key
11) Search for...
12) Sort the list
13) Print the elements in reverse
14) Reverse the list
15) Print nth node from end
16) Delete all duplicate items
17) Drop the list
18) Exit
->: 8

Enter the position at which to insert (-1 to insert at end): 2
Enter value to insert: 22
---------------------------
Enter your choice:
1) Create a new node
2) Display the list
3) Is the list empty?
4) Insert the node after...
5) Insert the node before...
6) Delete node after...
7) Delete node before...
8) Insert at position
9) Delete at position
10) Delete a node for the given key
11) Search for...
12) Sort the list
13) Print the elements in reverse
14) Reverse the list
```

```
15) Print nth node from end
16) Delete all duplicate items
17) Drop the list
18) Exit
->: 2

10->20->22->30->null
---------------------------
Enter your choice:
1) Create a new node
2) Display the list
3) Is the list empty?
4) Insert the node after...
5) Insert the node before...
6) Delete node after...
7) Delete node before...
8) Insert at position
9) Delete at position
10) Delete a node for the given key
11) Search for...
12) Sort the list
13) Print the elements in reverse
14) Reverse the list
15) Print nth node from end
16) Delete all duplicate items
17) Drop the list
18) Exit
->: 9
Enter the position to delete at (-1 to delete from end): 2
---------------------------
Enter your choice:
1) Create a new node
2) Display the list
3) Is the list empty?
4) Insert the node after...
5) Insert the node before...
6) Delete node after...
7) Delete node before...
8) Insert at position
9) Delete at position
10) Delete a node for the given key
11) Search for...
12) Sort the list
13) Print the elements in reverse
14) Reverse the list
15) Print nth node from end
16) Delete all duplicate items
17) Drop the list
18) Exit
->: 2
```

```
10->20->30->null
---------------------------
Enter your choice:
1) Create a new node
2) Display the list
3) Is the list empty?
4) Insert the node after...
5) Insert the node before...
6) Delete node after...
7) Delete node before...
8) Insert at position
9) Delete at position
10) Delete a node for the given key
11) Search for...
12) Sort the list
13) Print the elements in reverse
14) Reverse the list
15) Print nth node from end
16) Delete all duplicate items
17) Drop the list
18) Exit
->: 11

Search for element: 10
Element found at position(s): 0->null
---------------------------
Enter your choice:
1) Create a new node
2) Display the list
3) Is the list empty?
4) Insert the node after...
5) Insert the node before...
6) Delete node after...
7) Delete node before...
8) Insert at position
9) Delete at position
10) Delete a node for the given key
11) Search for...
12) Sort the list
13) Print the elements in reverse
14) Reverse the list
15) Print nth node from end
16) Delete all duplicate items
17) Drop the list
18) Exit
->: 12

The list was sorted!
---------------------------
```

```
Enter your choice:
1) Create a new node
2) Display the list
3) Is the list empty?
4) Insert the node after...
5) Insert the node before...
6) Delete node after...
7) Delete node before...
8) Insert at position
9) Delete at position
10) Delete a node for the given key
11) Search for...
12) Sort the list
13) Print the elements in reverse
14) Reverse the list
15) Print nth node from end
16) Delete all duplicate items
17) Drop the list
18) Exit
->: 2

10->20->30->null
---------------------------
Enter your choice:
1) Create a new node
2) Display the list
3) Is the list empty?
4) Insert the node after...
5) Insert the node before...
6) Delete node after...
7) Delete node before...
8) Insert at position
9) Delete at position
10) Delete a node for the given key
11) Search for...
12) Sort the list
13) Print the elements in reverse
14) Reverse the list
15) Print nth node from end
16) Delete all duplicate items
17) Drop the list
18) Exit
->: 13

null<-30<-20<-10 -
---------------------------
Enter your choice:
1) Create a new node
2) Display the list
3) Is the list empty?
```

```
4) Insert the node after...
5) Insert the node before...
6) Delete node after...
7) Delete node before...
8) Insert at position
9) Delete at position
10) Delete a node for the given key
11) Search for...
12) Sort the list
13) Print the elements in reverse
14) Reverse the list
15) Print nth node from end
16) Delete all duplicate items
17) Drop the list
18) Exit
->: 14

The list was reversed!
---------------------------
Enter your choice:
1) Create a new node
2) Display the list
3) Is the list empty?
4) Insert the node after...
5) Insert the node before...
6) Delete node after...
7) Delete node before...
8) Insert at position
9) Delete at position
10) Delete a node for the given key
11) Search for...
12) Sort the list
13) Print the elements in reverse
14) Reverse the list
15) Print nth node from end
16) Delete all duplicate items
17) Drop the list
18) Exit
->: 2

30->20->10->null
---------------------------
Enter your choice:
1) Create a new node
2) Display the list
3) Is the list empty?
4) Insert the node after...
5) Insert the node before...
6) Delete node after...
7) Delete node before...
```

```
8) Insert at position
9) Delete at position
10) Delete a node for the given key
11) Search for...
12) Sort the list
13) Print the elements in reverse
14) Reverse the list
15) Print nth node from end
16) Delete all duplicate items
17) Drop the list
18) Exit
->: 15

Enter node number: 2
The node no. 2 from the end is: 20
---------------------------
Enter your choice:
1) Create a new node
2) Display the list
3) Is the list empty?
4) Insert the node after...
5) Insert the node before...
6) Delete node after...
7) Delete node before...
8) Insert at position
9) Delete at position
10) Delete a node for the given key
11) Search for...
12) Sort the list
13) Print the elements in reverse
14) Reverse the list
15) Print nth node from end
16) Delete all duplicate items
17) Drop the list
18) Exit
->: 1

Enter an element: 10
---------------------------
Enter your choice:
1) Create a new node
2) Display the list
3) Is the list empty?
4) Insert the node after...
5) Insert the node before...
6) Delete node after...
7) Delete node before...
8) Insert at position
9) Delete at position
10) Delete a node for the given key
```

```
11) Search for...
12) Sort the list
13) Print the elements in reverse
14) Reverse the list
15) Print nth node from end
16) Delete all duplicate items
17) Drop the list
18) Exit
->: 2

30->20->10->10->null
---------------------------
Enter your choice:
1) Create a new node
2) Display the list
3) Is the list empty?
4) Insert the node after...
5) Insert the node before...
6) Delete node after...
7) Delete node before...
8) Insert at position
9) Delete at position
10) Delete a node for the given key
11) Search for...
12) Sort the list
13) Print the elements in reverse
14) Reverse the list
15) Print nth node from end
16) Delete all duplicate items
17) Drop the list
18) Exit
->: 16

All list elements are now unique!
---------------------------
Enter your choice:
1) Create a new node
2) Display the list
3) Is the list empty?
4) Insert the node after...
5) Insert the node before...
6) Delete node after...
7) Delete node before...
8) Insert at position
9) Delete at position
10) Delete a node for the given key
11) Search for...
12) Sort the list
13) Print the elements in reverse
14) Reverse the list
```

```
15) Print nth node from end
16) Delete all duplicate items
17) Drop the list
18) Exit
->: 2

30->20->10->null
---------------------------
Enter your choice:
1) Create a new node
2) Display the list
3) Is the list empty?
4) Insert the node after...
5) Insert the node before...
6) Delete node after...
7) Delete node before...
8) Insert at position
9) Delete at position
10) Delete a node for the given key
11) Search for...
12) Sort the list
13) Print the elements in reverse
14) Reverse the list
15) Print nth node from end
16) Delete all duplicate items
17) Drop the list
18) Exit
->: 17

List cleared!
---------------------------
Enter your choice:
1) Create a new node
2) Display the list
3) Is the list empty?
4) Insert the node after...
5) Insert the node before...
6) Delete node after...
7) Delete node before...
8) Insert at position
9) Delete at position
10) Delete a node for the given key
11) Search for...
12) Sort the list
13) Print the elements in reverse
14) Reverse the list
15) Print nth node from end
16) Delete all duplicate items
17) Drop the list
18) Exit
```

```
->: 2

null
---------------------------
Enter your choice:
1) Create a new node
2) Display the list
3) Is the list empty?
4) Insert the node after...
5) Insert the node before...
6) Delete node after...
7) Delete node before...
8) Insert at position
9) Delete at position
10) Delete a node for the given key
11) Search for...
12) Sort the list
13) Print the elements in reverse
14) Reverse the list
15) Print nth node from end
16) Delete all duplicate items
17) Drop the list
18) Exit
->: 18

Exiting...
---------------------------
```