

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

struct node {
    int data;
    int prev;
    struct node *next;
};

struct node *head = NULL;

void createList(int n) {
    struct node *newNode, *temp = NULL;
    int data, i;

    if (n <= 0) {
        printf("Number of nodes should be greater than 0\n");
        return;
    }

    if (head != NULL) {
        printf("Warning: Overwriting existing list.\n");
        head = NULL;
    }

    for (i = 1; i <= n; i++) {
        newNode = (struct node*)malloc(sizeof(struct node));
        if (newNode == NULL) {
            printf("Memory allocation failed\n");
            return;
        }

        printf("Enter data for node %d: ", i);
        scanf("%d", &data);

        newNode->data = data;
        newNode->next = NULL;

        if (head == NULL)
            head = newNode;
        else
            temp->next = newNode;

        temp = newNode;
    }

    printf("\nlinked list created successfully\n");
}

void displayList() {
    struct node *temp = head;
}
```

```
void displaylist() {
    struct node *temp = head;
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }

    printf("\nlinked list: ");
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

void deleteBeginning() {
    struct node *temp;
    if (head == NULL) {
        printf("List is empty");
        return;
    }

    temp = head;
    head = head->next;
    temp->data;
    free (temp);
}

void deleteatend()
{
    struct node *temp, *prev;
    if(head==NULL)
        printf("List is empty");
    return;
}

if(head->next==NULL)
    printf("Deleted element:%d\n",head->data);
head=NULL;
return;

temp = head;
while(temp->next!=NULL)
    prev=temp;
    temp=temp->next;
}

printf("Deleted element:%d\n",temp->data);
prev->next=NULL;
free (temp);
}

void deletepos(int pos) {
    if (head == NULL) {
        printf("List is empty\n");
    }
```

```
 singlydeletion.c - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Plugins DocBlocks Settings Help
Start here X doublylinkedlist X Sort_Reverse_Concat X BSTc X singlydeletion.c X
103 void deleteatpos(int pos) {
104     if (head == NULL) {
105         printf("List is empty\n");
106         return;
107     }
108
109     if (pos < 0) {
110         printf("Invalid position\n");
111         return;
112     }
113
114     struct node *temp = head, *prev = NULL;
115     int count = 1;
116
117     if (pos == 1) {
118         head = head->next;
119         print("Deleted element: %d\n", temp->data);
120         free(temp);
121         return;
122     }
123     while (temp != NULL && count < pos) {
124         prev = temp;
125         temp = temp->next;
126         count++;
127     }
128
129     if (temp == NULL) {
130         printf("Position out of range\n");
131         return;
132     }
133
134     prev->next = temp->next;
135     printf("Deleted element: %d\n", temp->data);
136     free(temp);
137 }
138
139
140
141
142 int main() {
143     int choice, n, data, pos;
144
145     while (1) {
146         printf("\n---- Singly Linked List Operations ----\n");
147         printf("1. Create linked list\n");
148         printf("2. Delete at beginning\n");
149         printf("3. Delete at end\n");
150         printf("4. Delete at pos\n");
151         printf("5. Display list\n");
152         printf("6. Exit\n");
153         printf("Enter your choice: ");
154         scanf("%d", &choice);
155         if (scanf("%d", &choice) != 1) {
156
157             while (getchar() != '\n')
158                 printf("Invalid input. Please enter a number.\n");
159             continue;
160         }
161
162         switch (choice) {
163             case 1:
164                 printf("Enter number of nodes: ");
165                 scanf("%d", &n);
166                 createList(n);
167                 break;
168             case 2:
169                 deleteatbeginning();
170                 break;
171             case 3:
172                 deleteatend();
173                 break;
174             case 4:
175                 printf("Enter position to delete: ");
176                 scanf("%d", &pos);
177                 deleteatpos(pos);
178                 break;
179
180             case 5:
181                 displayList();
182                 break;
183
184             case 6:
185                 printf("Exiting...\n");
186                 exit(0);
187             default:
188                 printf("Invalid choice. Try again.\n");
189         }
190     }
191
192
193
194
195 }
```

```
 singlydeletion.c - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Plugins DocBlocks Settings Help
Start here X doublylinkedlist X Sort_Reverse_Concat X BSTc X singlydeletion.c X
142
143 int main() {
144     int choice, n, data, pos;
145
146     while (1) {
147         printf("\n---- Singly Linked List Operations ----\n");
148         printf("1. Create linked list\n");
149         printf("2. Delete at beginning\n");
150         printf("3. Delete at end\n");
151         printf("4. Delete at pos\n");
152         printf("5. Display list\n");
153         printf("6. Exit\n");
154         printf("Enter your choice: ");
155         if (scanf("%d", &choice) != 1) {
156
157             while (getchar() != '\n')
158                 printf("Invalid input. Please enter a number.\n");
159             continue;
160         }
161
162         switch (choice) {
163             case 1:
164                 printf("Enter number of nodes: ");
165                 scanf("%d", &n);
166                 createList(n);
167                 break;
168             case 2:
169                 deleteatbeginning();
170                 break;
171             case 3:
172                 deleteatend();
173                 break;
174             case 4:
175                 printf("Enter position to delete: ");
176                 scanf("%d", &pos);
177                 deleteatpos(pos);
178                 break;
179
180             case 5:
181                 displayList();
182                 break;
183
184             case 6:
185                 printf("Exiting...\n");
186                 exit(0);
187             default:
188                 printf("Invalid choice. Try again.\n");
189         }
190     }
191
192
193
194
195 }
```

```

chethan01@chethan-OptiPlex-5090:~/Desktop$ ./SinglyListDelete.exe
1. insert at beginning
2. delete at beginning
3. delete at end
4. delete at pos
5. display list
6. Exit
Enter your choice: 1
Enter number of nodes: 3
Enter data for node 1: 23
Enter data for node 2: 45
Enter data for node 3: 46
Linked list created successfully
---- Singly Linked List Operations ----
1. Create linked list
2. Delete at beginning
3. delete at end
4. delete at pos
5. Display list
6. Exit
Enter your choice: 5
Linked list: 23 -> 45 -> 46 -> NULL
---- Singly Linked List Operations ----
1. Create linked list
2. Delete at beginning
3. delete at end
4. delete at pos
5. Display list
6. Exit
Enter your choice: 2
Enter position to delete: 2
Deleted element: 45
Linked list: 23 -> NULL
---- Singly Linked List Operations ----
1. Create linked list
2. Delete at beginning
3. delete at end
4. delete at pos
5. Display list
6. Exit
Enter your choice: 5
Linked list: 23 -> NULL
---- Singly Linked List Operations ----
1. Create linked list
2. Delete at beginning
3. delete at end
4. delete at pos
5. Display list
6. Exit
Enter your choice: 6
Enter position to delete: 1
Deleted element: 23
Linked list: NULL
---- Singly Linked List Operations ----
1. Create linked list
2. Delete at beginning
3. delete at end
4. delete at pos
5. Display list
6. Exit
Enter your choice: 5

```

leetcode.com/problems/remove-linked-list-elements/

**203. Remove Linked List Elements**

Given the head of a linked list and an integer val, remove all the nodes of the linked list that has Node.val == val, and return the new head.

**Example 1:**

**Input:** head = [1,2,6,3,4,5,6], val = 6  
**Output:** [1,2,3,4,5]

**Example 2:**

**Input:** head = [], val = 1  
**Output:** []

**Example 3:**

**Input:** head = [7,7,7,7], val = 7  
**Output:** []

**Code:**

```

1 struct ListNode* removeElements(struct ListNode* head, int val) {
2     struct ListNode dummy;
3     dummy.next = head;
4     struct ListNode* curr = &dummy;
5     while (curr->next != NULL) {
6         if (curr->next->val == val) {
7             curr->next = curr->next->next;
8         } else {
9             curr = curr->next;
10        }
11    }
12    return dummy.next;
}

```

**Accepted** 66 / 66 testcases passed

chethanmbse25 submitted at Nov 20, 2025 18:41

**Runtime**  
0 ms | Beats 100.00% ⚡  
@ Memory  
12.72 MB | Beats 16.90%

Code | C

Testcase | Test Result

22°C Partly cloudy 20:57 08-12-2025

Implement singly linked list following operations of reverse linked list

b) Deletion of first element, specified element and last element

c) display.

pseudocode:

```
node {
```

```
    data
```

```
    }
```

```
head
```

Delete from Beginning

deleteBeginning ():

if head == NULL;

print ("list is empty")

endif

temp = head

head = head.next

free temp

Delete at end

deleteEnd ():

if head == NULL;

print ("list is empty")

return

if head.next == NULL;

free head

head = null

return

current = head

while current.next != NULL;

current = current.next



Shot on OnePlus

chE\_Reddy

Free current.next  
current.next = null  
end function

deletWithValue(value):

if head == null:

print "List is empty"

return

endif

if head.data == value:

temp = head

head = head.next

Free temp

return

end

current = head

while current.next != null and current.next.data != value:

current = current.next

end

if current.next == null:

print value not found

end

temp = current.next

current.next = current.next.next

Free temp

end

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

struct node {
    int data;
    int prev;
    struct node *next;
};

struct node *head = NULL;

void createList(int n) {
    struct node *newnode, *temp = NULL;
    int data;
    if (n <= 0) {
        printf("number of nodes should be greater than 0");
        return;
    }
    if (head == NULL) {
        printf("Warning: overwriting existing list.\n");
        head = NULL;
    }
    for (i=1; i<=n; i++) {
        newnode = (struct node *) malloc(sizeof(struct node));
        if (newnode == NULL) {
            printf("Memory allocation failed.\n");
            return;
        }
        printf("Enter data for node %d : ", i);
        scanf("%d", &data);
        newnode->data = data;
        newnode->next = NULL;
```

```
if(head == NULL)
    head = newnode;
else
    temp->next = newnode;
    temp = newnode;
}

void displaylist () {
    struct node *temp = head;
    if(head == NULL){
        printf("list is empty\n");
        return;
    }
    printf("In linked list:\n");
    while ( temp != NULL){
        printf("%d ->", temp->data);
        temp = temp->next;
    }
    printf("\nnull\n");
}
```

```
void deleteatbeginning () {
    struct node *temp;
    if (head == NULL) {
        printf("list is empty");
        return;
    }
}
```

```
temp = head;
head = head->next;
free (temp);
```

```

void deleteend() {
    struct node* temp, * prev;
    if (head == NULL) {
        printf("list is empty");
        return;
    }
    void deletpos(int pos) {
        if (head == NULL) {
            printf("list is empty\n");
            return;
        }
        if (pos <= 0)
            printf("invalid position\n");
        return;
    }
    struct node* temp = head, * prev = NULL;
    int count = 1;
    if (pos == 1) {
        head = head->next;
        printf("deleted element: %d\n", temp->data);
        free(temp);
        return;
    }
    while (temp != NULL && count < pos) {
        prev = temp;
        temp = temp->next;
        count++;
    }
    if (temp == NULL) {
        printf("position out of range\n");
        return;
    }
}

```

```

    pren->next = temp -> next;
    prints (" deleted element: " <temp> data);
    free (temp);
}

int main() {
    int choice, n, data, pos;
    while (1) {
        printf ("1. ... singly linked list ... (\n");
        printf ("2. create a linked list (\n");
        printf ("3. delete at beginning (\n");
        printf ("4. delete at end (\n");
        printf ("5. delete at pos (\n");
        printf ("6. Display list (\n");
        printf ("7. Exist (\n");
        printf ("Enter your choice:");
        if (scanf ("%d", &choice) != 1) {
            while (getchar () != '\n');
            printf ("Invalid input. please enter a number. (\n");
            continue;
        }
        switch (choice) {
            case 1:
                printf ("Enter number of nodes: ");
                scanf ("%d", &n);
                createlist (n);
                break;
            case 2:
                deleteatbeginning ();
                break;
            case 3:
                deleteatend ();
                break;
        }
    }
}

```

CASE 4:

```
printf("Enter position to delete");
scanf("%d", &pos);
deletepos(pos);
break;
```

CASE 5:

```
displaylist();
break;
```

CASE 6:

```
printf("Exiting... In");
exit(0);
```

default:

```
printf("Invalid choice. Try Again.\n");
```

}

Output:

singly linkedlist

1. Create a linked list
2. Delete at beginning
3. Delete at end
4. Delete at pos.
5. Display list
6. Exit!

Enter your choice: 1

Enter number of nodes: 4

Enter data for node: 2

Enter data for node: 4

Enter data for node: 6

Enter data for node: 8

Enter your choice: 5  
2 -> 4 -> 6 -> 8 -> NULL

Enter your choice: 8  
Deleted element: 8

Enter your choice: 5  
2 -> 4 -> 6 -> 8 -> NULL

Enter your choice: 4  
Enter your position to delete: 2  
Deleted element: 6

Enter your choice: 3  
4 -> 8 -> NULL

Enter your choice: 3  
Deleted element: 8

20/11/25

### Remove linked list elements

Code:

```
struct ListNode * RemoveElements (struct ListNode * head, int val) {  
    struct ListNode dummy;  
    dummy.next = head;  
    struct ListNode * prev = &dummy;  
    while (prev->next != NULL) {  
        if (prev->next->val == val) {  
            struct ListNode * toDelete = prev->next;  
            prev->next = toDelete->next;  
            free (toDelete);  
        } else {  
            prev = prev->next;  
        }  
    }  
    return dummy.next;
```