

## 1. Deletion at beginning

Programiz  
C Online Compiler

Capgemini  
Rewrite your future.  
Join us.

main.c

1 #include <stdio.h>  
2 #include <stdlib.h>  
3 struct Node {  
4 int data;  
5 struct Node \*next;  
6 };  
7 void deleteAtStart(struct Node \*\*head) {  
8 if (\*head == NULL) {  
9 printf("List is empty\n");  
10 return;  
11 }  
12 struct Node \*temp = \*head;  
13 \*head = (\*head)->next;  
14 free(temp);  
15 }  
16 void display(struct Node \*head) {  
17 struct Node \*temp = head;  
18 while (temp != NULL) {  
19 printf("%d -> ", temp->data);  
20 temp = temp->next;  
21 }  
22 printf("NULL\n");  
23 }  
24 int main() {  
25 struct Node \*head, \*first, \*second, \*third;  
26 head = (struct Node\*)malloc(sizeof(struct Node));  
27 first = (struct Node\*)malloc(sizeof(struct Node));  
28 second = (struct Node\*)malloc(sizeof(struct Node));  
29 }

Run

Output

Clear

Original List:  
20 -> 40 -> 70 -> 65 -> NULL  
After deleting first node:  
40 -> 70 -> 65 -> NULL  
  
=== Code Execution Successful ===

## 2. Deletion at end



main.c

Share

Run

Output

Clear

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 struct Node {
4     int data;
5     struct Node *next;
6 };
7 void deleteAtPosition(struct Node **head, int position) {
8     if (*head == NULL) {
9         printf("List is empty\n");
10        return;
11    }
12    struct Node *temp = *head;
13    if (position == 1) {
14        *head = temp->next;
15        free(temp);
16        return;
17    }
18    for (int i = 1; temp != NULL && i < position - 1; i++) {
19        temp = temp->next;
20    }
21    if (temp == NULL || temp->next == NULL) {
22        printf("Invalid position\n");
23        return;
24    }
25    struct Node *nodeToDelete = temp->next;
26    temp->next = nodeToDelete->next;
27    free(nodeToDelete);
28 }

```

```

Original List:
40 -> 20 -> 150 -> 180 -> NULL
After deleting node at position 3:
40 -> 20 -> 180 -> NULL

=== Code Execution Successful ===

```

main.c

Share

Run

Output

Clear

```

28 }
29 void display(struct Node *head) {
30     struct Node *temp = head;
31     while (temp != NULL) {
32         printf("%d -> ", temp->data);
33         temp = temp->next;
34     }
35     printf("NULL\n");
36 }
37 int main() {
38     struct Node *head, *first, *second, *third;
39     head = (struct Node*)malloc(sizeof(struct Node));
40     first = (struct Node*)malloc(sizeof(struct Node));
41     second = (struct Node*)malloc(sizeof(struct Node));
42     third = (struct Node*)malloc(sizeof(struct Node));
43     head->data = 40;
44     head->next = first;
45     first->data = 20;
46     first->next = second;
47     second->data = 150;
48     second->next = third;
49     third->data = 180;
50     third->next = NULL;
51     printf("Original List:\n");
52     display(head);
53     int position = 3;
54     deleteAtPosition(&head, position);
55     printf("After deleting node at position %d:\n", position);

```

```

Original List:
40 -> 20 -> 150 -> 180 -> NULL
After deleting node at position 3:
40 -> 20 -> 180 -> NULL

=== Code Execution Successful ===

```

main.c

Run

Share

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 struct Node {
4     int data;
5     struct Node *next;
6 };
7 void deleteAtPosition(struct Node **head, int position) {
8     if (*head == NULL) {
9         printf("List is empty\n");
10        return;
11    }
12    struct Node *temp = *head;
13    if (position == 1) {
14        *head = temp->next;
15        free(temp);
16        return;
17    }
18    for (int i = 1; temp != NULL && i < position - 1; i++) {
19        temp = temp->next;
20    }
21    if (temp == NULL || temp->next == NULL) {
22        printf("Invalid position\n");
23        return;
24    }
25    struct Node *nodeToDelete = temp->next;
26    temp->next = nodeToDelete->next;
27    free(nodeToDelete);
28 }
```

Output

Clear

Original List:  
40 -> 20 -> 150 -> 180 -> NULL  
After deleting node at position 3:  
40 -> 20 -> 180 -> NULL  
  
=== Code Execution Successful ===