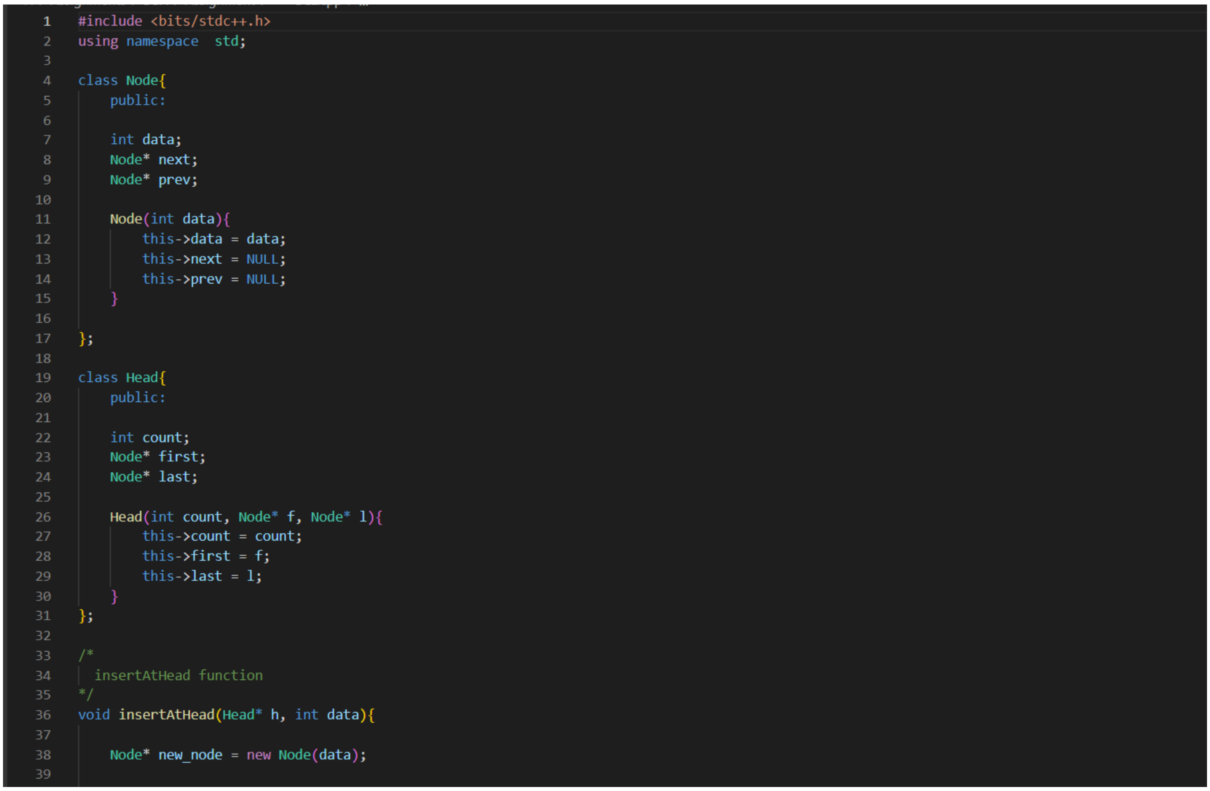
PRN: 2020BTEIT00041

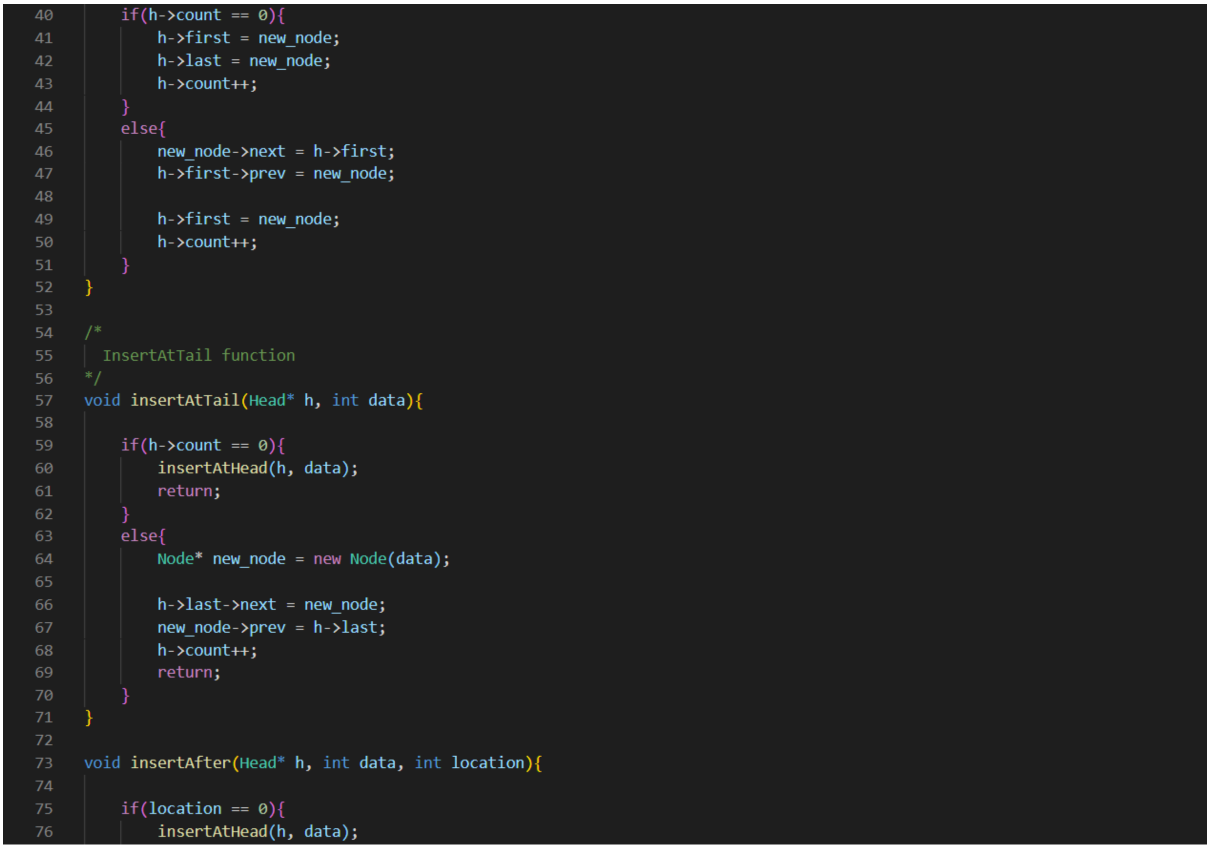
NAME: Om Vivek Gharge

Q.) Doubly Linked List:

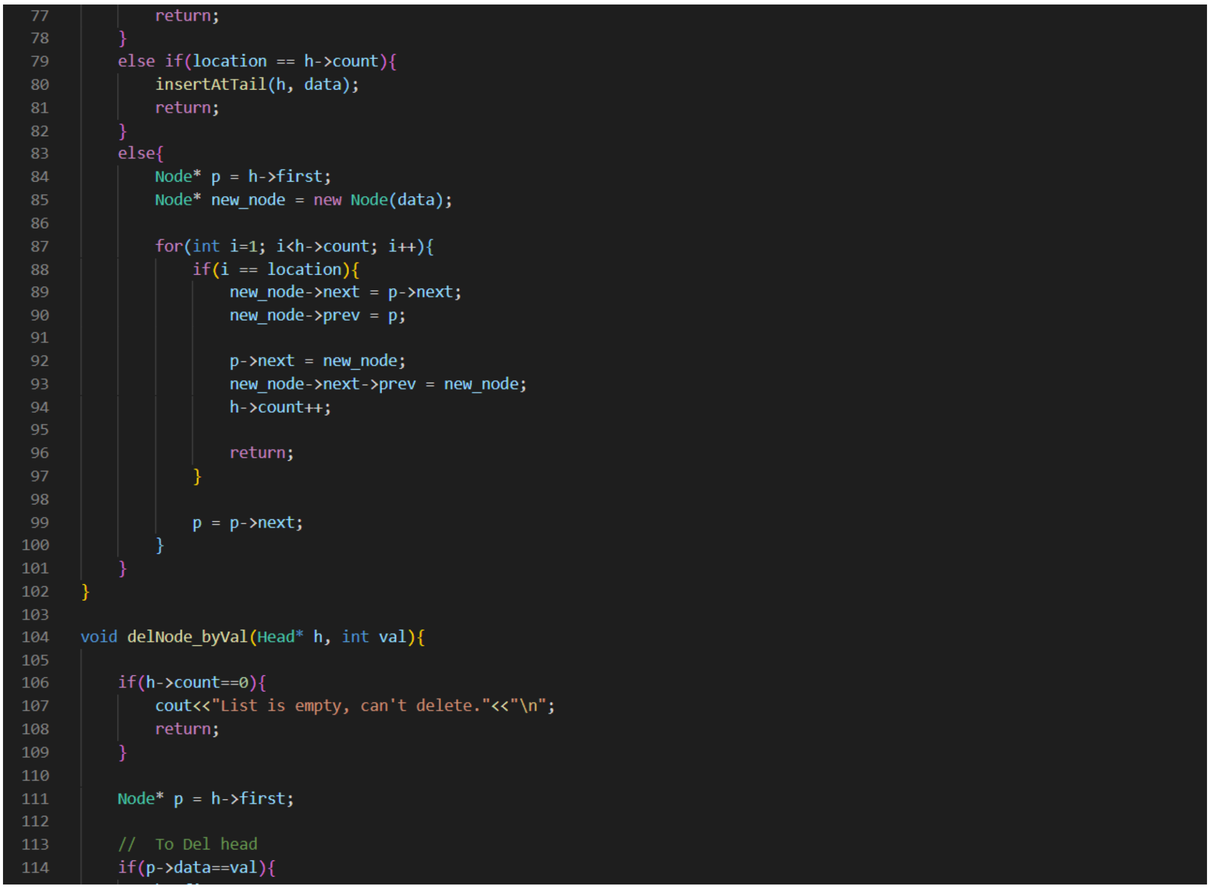
CODE:



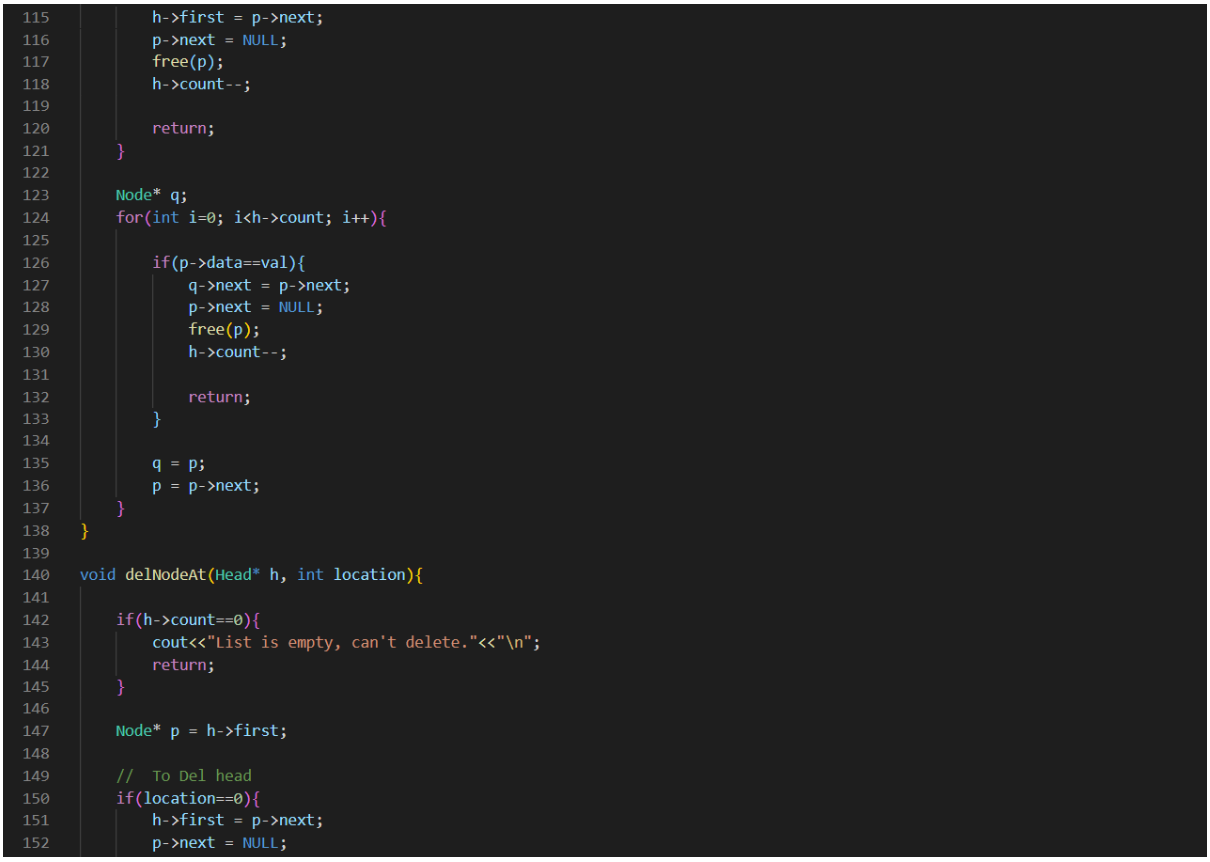
```cpp
#include <bits/stdc++.h>
using namespace  std;

class Node{
    public:

    int data;
    Node* next;
    Node* prev;

    Node(int data){
        this->data = data;
        this->next = NULL;
        this->prev = NULL;
    }

};

class Head{
    public:

    int count;
    Node* first;
    Node* last;

    Head(int count, Node* f, Node* l){
        this->count = count;
        this->first = f;
        this->last = l;
    }
};

/*
  insertAtHead function
*/
void insertAtHead(Head* h, int data){

    Node* new_node = new Node(data);

```



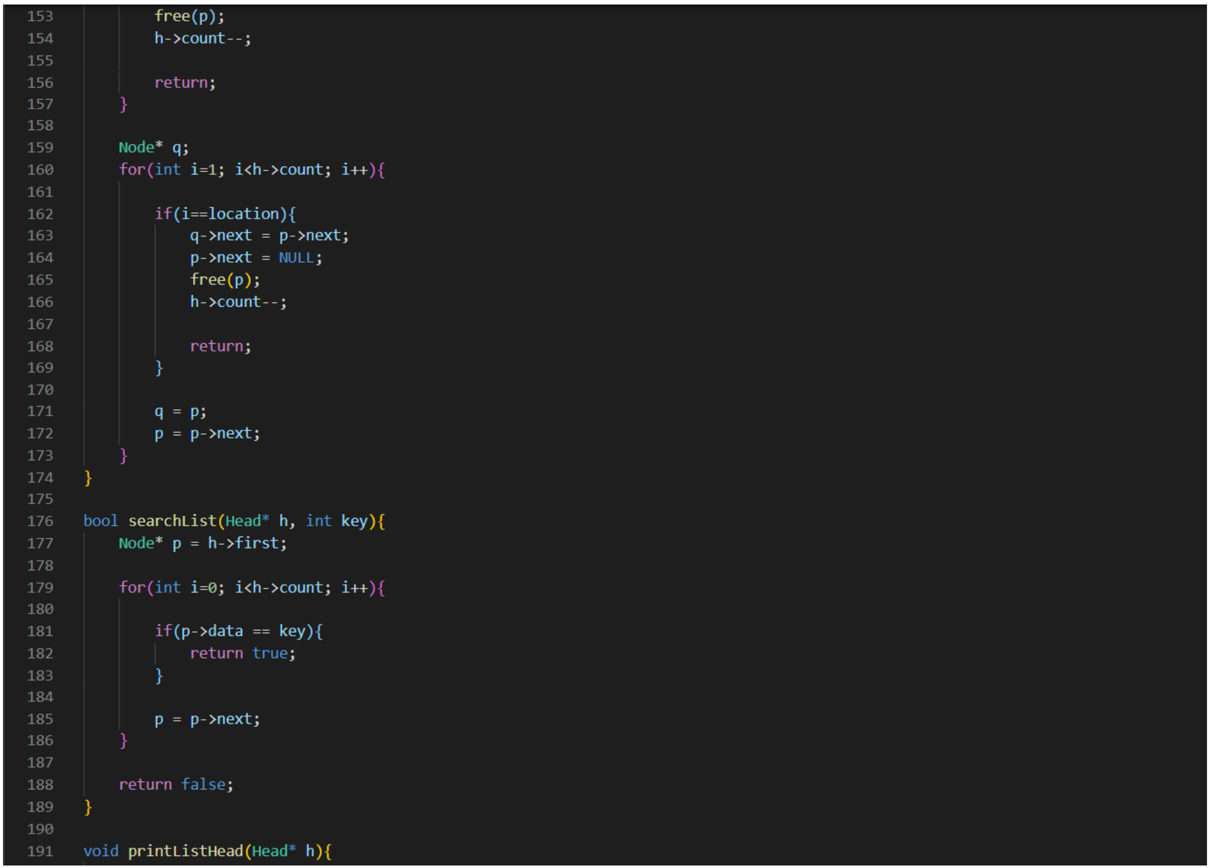
```cpp
    if(h->count == 0){
        h->first = new_node;
        h->last = new_node;
        h->count++;
    }
    else{
        new_node->next = h->first;
        h->first->prev = new_node;

        h->first = new_node;
        h->count++;
    }
}

/*
  InsertAtTail function
*/
void insertAtTail(Head* h, int data){

    if(h->count == 0){
        insertAtHead(h, data);
        return;
    }
    else{
        Node* new_node = new Node(data);

        h->last->next = new_node;
        new_node->prev = h->last;
        h->count++;
        return;
    }
}

void insertAfter(Head* h, int data, int location){

    if(location == 0){
        insertAtHead(h, data);
```



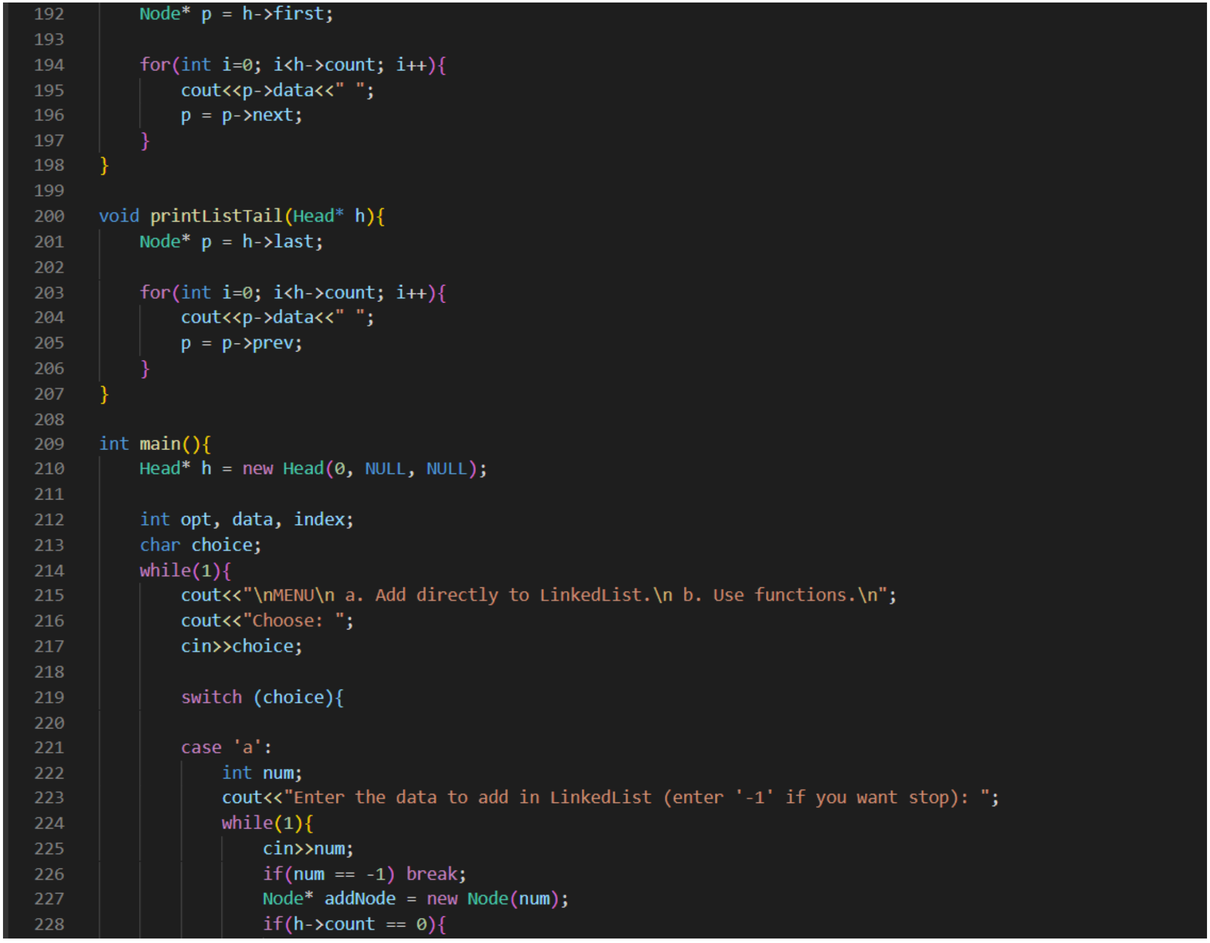
```
        return;
    }
    else if(location == h->count){
        insertAtTail(h, data);
        return;
    }
    else{
        Node* p = h->first;
        Node* new_node = new Node(data);

        for(int i=1; i<h->count; i++){
            if(i == location){
                new_node->next = p->next;
                new_node->prev = p;

                p->next = new_node;
                new_node->next->prev = new_node;
                h->count++;

                return;
            }

            p = p->next;
        }
    }
}

void delNode_byVal(Head* h, int val){

    if(h->count==0){
        cout<<"List is empty, can't delete."<<"\n";
        return;
    }

    Node* p = h->first;

    //  To Del head
    if(p->data==val){
```



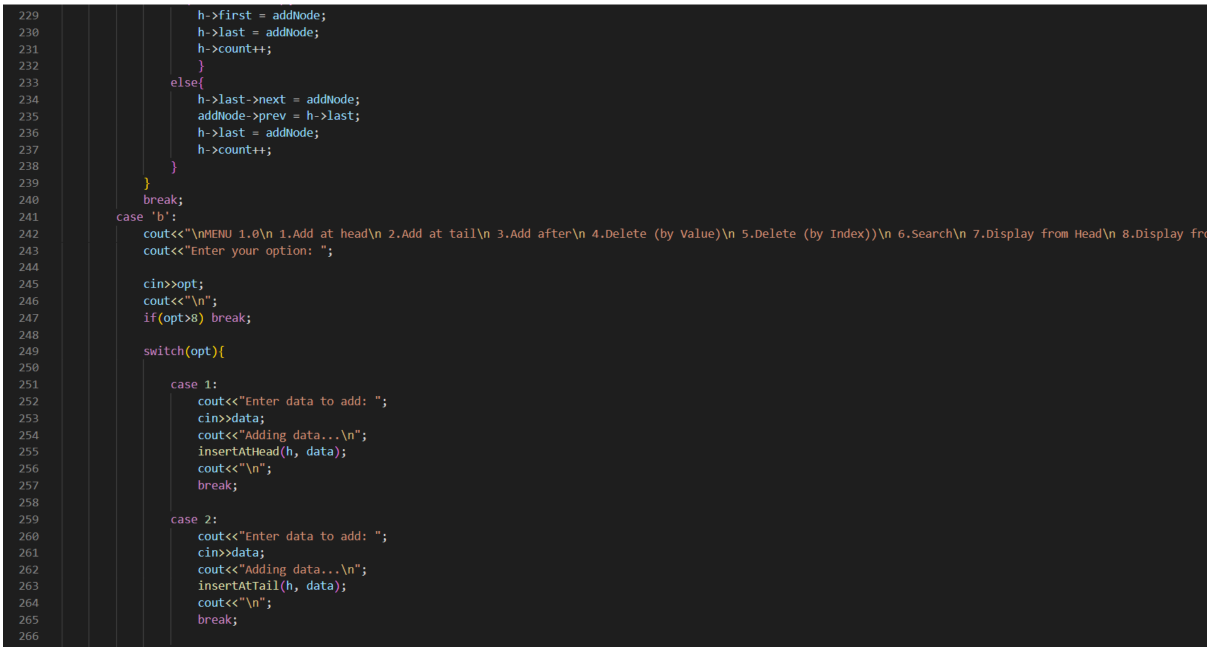
```
        h->first = p->next;
        p->next = NULL;
        free(p);
        h->count--;

        return;
    }

    Node* q;
    for(int i=0; i<h->count; i++){

        if(p->data==val){
            q->next = p->next;
            p->next = NULL;
            free(p);
            h->count--;

            return;
        }

        q = p;
        p = p->next;
    }
}

void delNodeAt(Head* h, int location){

    if(h->count==0){
        cout<<"List is empty, can't delete."<<"\n";
        return;
    }

    Node* p = h->first;

    //  To Del head
    if(location==0){
        h->first = p->next;
        p->next = NULL;
```



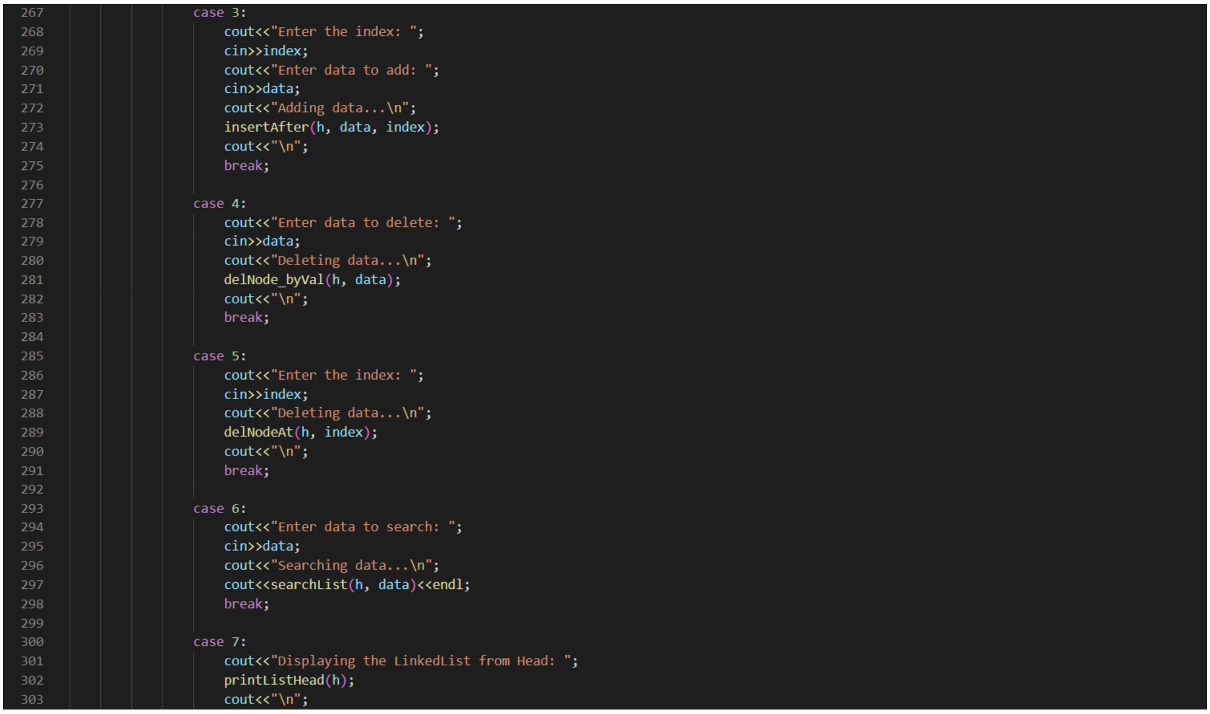
```
        free(p);
        h->count--;

        return;
    }

    Node* q;
    for(int i=1; i<h->count; i++){

        if(i==location){
            q->next = p->next;
            p->next = NULL;
            free(p);
            h->count--;

            return;
        }

        q = p;
        p = p->next;
    }
}

bool searchList(Head* h, int key){
    Node* p = h->first;

    for(int i=0; i<h->count; i++){

        if(p->data == key){
            return true;
        }

        p = p->next;
    }

    return false;
}

void printListHead(Head* h){
```



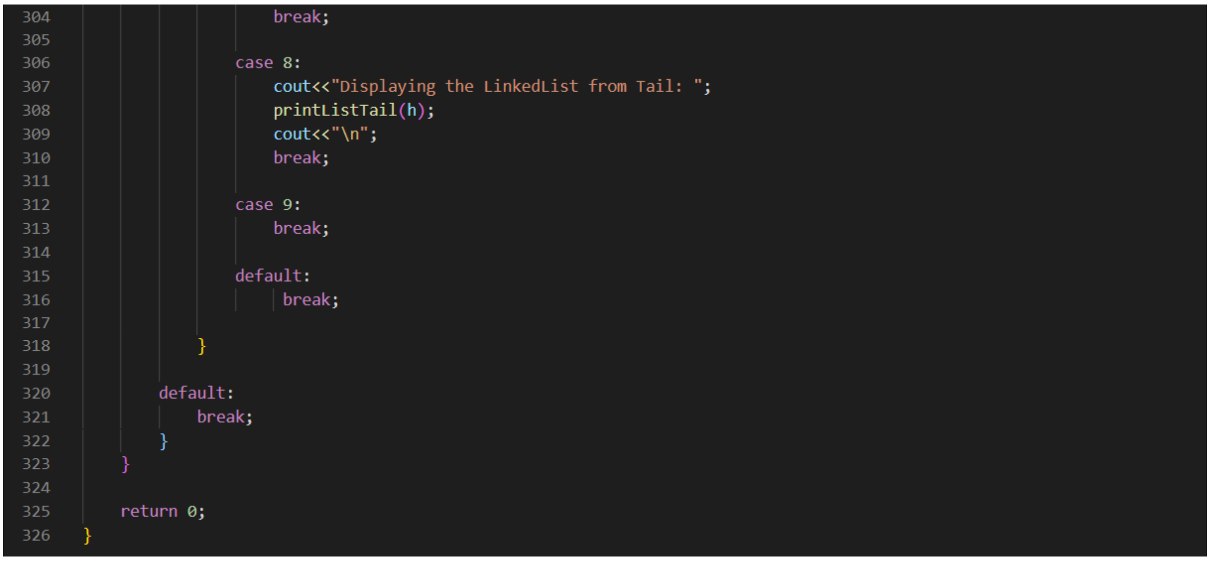
```
    Node* p = h->first;

    for(int i=0; i<h->count; i++){
        cout<<p->data<<" ";
        p = p->next;
    }
}

void printListTail(Head* h){
    Node* p = h->last;

    for(int i=0; i<h->count; i++){
        cout<<p->data<<" ";
        p = p->prev;
    }
}

int main(){
    Head* h = new Head(0, NULL, NULL);

    int opt, data, index;
    char choice;
    while(1){
        cout<<"\nMENU\n a. Add directly to LinkedList.\n b. Use functions.\n";
        cout<<"Choose: ";
        cin>>choice;

        switch (choice){

        case 'a':
            int num;
            cout<<"Enter the data to add in LinkedList (enter '-1' if you want stop): ";
            while(1){
                cin>>num;
                if(num == -1) break;
                Node* addNode = new Node(num);
                if(h->count == 0){
```



```
                    h->first = addNode;
                    h->last = addNode;
                    h->count++;
                    }
                else{
                    h->last->next = addNode;
                    addNode->prev = h->last;
                    h->last = addNode;
                    h->count++;
                }
            }
            break;
        case 'b':
            cout<<"\nMENU 1.0\n 1.Add at head\n 2.Add at tail\n 3.Add after\n 4.Delete (by Value)\n 5.Delete (by Index))\n 6.Search\n 7.Display from Head\n 8.Display fr
            cout<<"Enter your option: ";

            cin>>opt;
            cout<<"\n";
            if(opt>8) break;

            switch(opt){

                case 1:
                    cout<<"Enter data to add: ";
                    cin>>data;
                    cout<<"Adding data...\n";
                    insertAtHead(h, data);
                    cout<<"\n";
                    break;

                case 2:
                    cout<<"Enter data to add: ";
                    cin>>data;
                    cout<<"Adding data...\n";
                    insertAtTail(h, data);
                    cout<<"\n";
                    break;

```

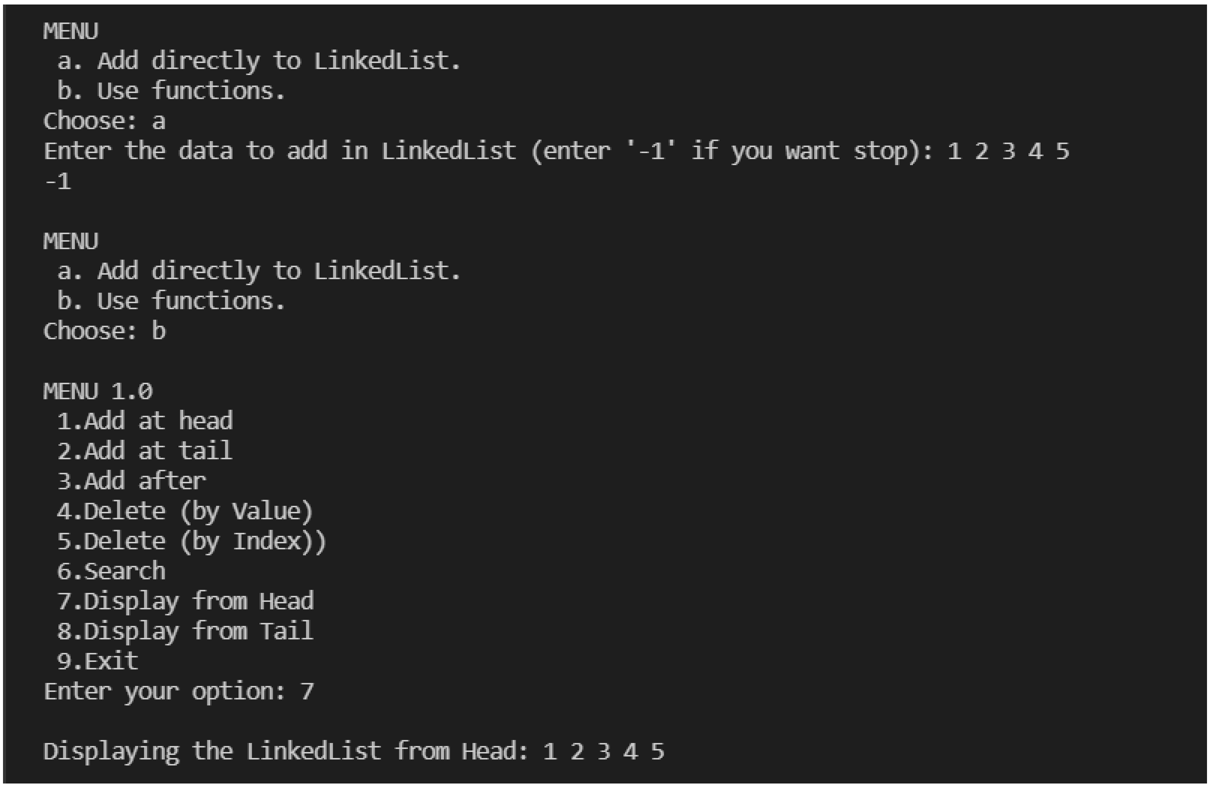


```
                case 3:
                    cout<<"Enter the index: ";
                    cin>>index;
                    cout<<"Enter data to add: ";
                    cin>>data;
                    cout<<"Adding data...\n";
                    insertAfter(h, data, index);
                    cout<<"\n";
                    break;

                case 4:
                    cout<<"Enter data to delete: ";
                    cin>>data;
                    cout<<"Deleting data...\n";
                    delNode_byVal(h, data);
                    cout<<"\n";
                    break;

                case 5:
                    cout<<"Enter the index: ";
                    cin>>index;
                    cout<<"Deleting data...\n";
                    delNodeAt(h, index);
                    cout<<"\n";
                    break;

                case 6:
                    cout<<"Enter data to search: ";
                    cin>>data;
                    cout<<"Searching data...\n";
                    cout<<searchList(h, data)<<endl;
                    break;

                case 7:
                    cout<<"Displaying the LinkedList from Head: ";
                    printListHead(h);
                    cout<<"\n";
```

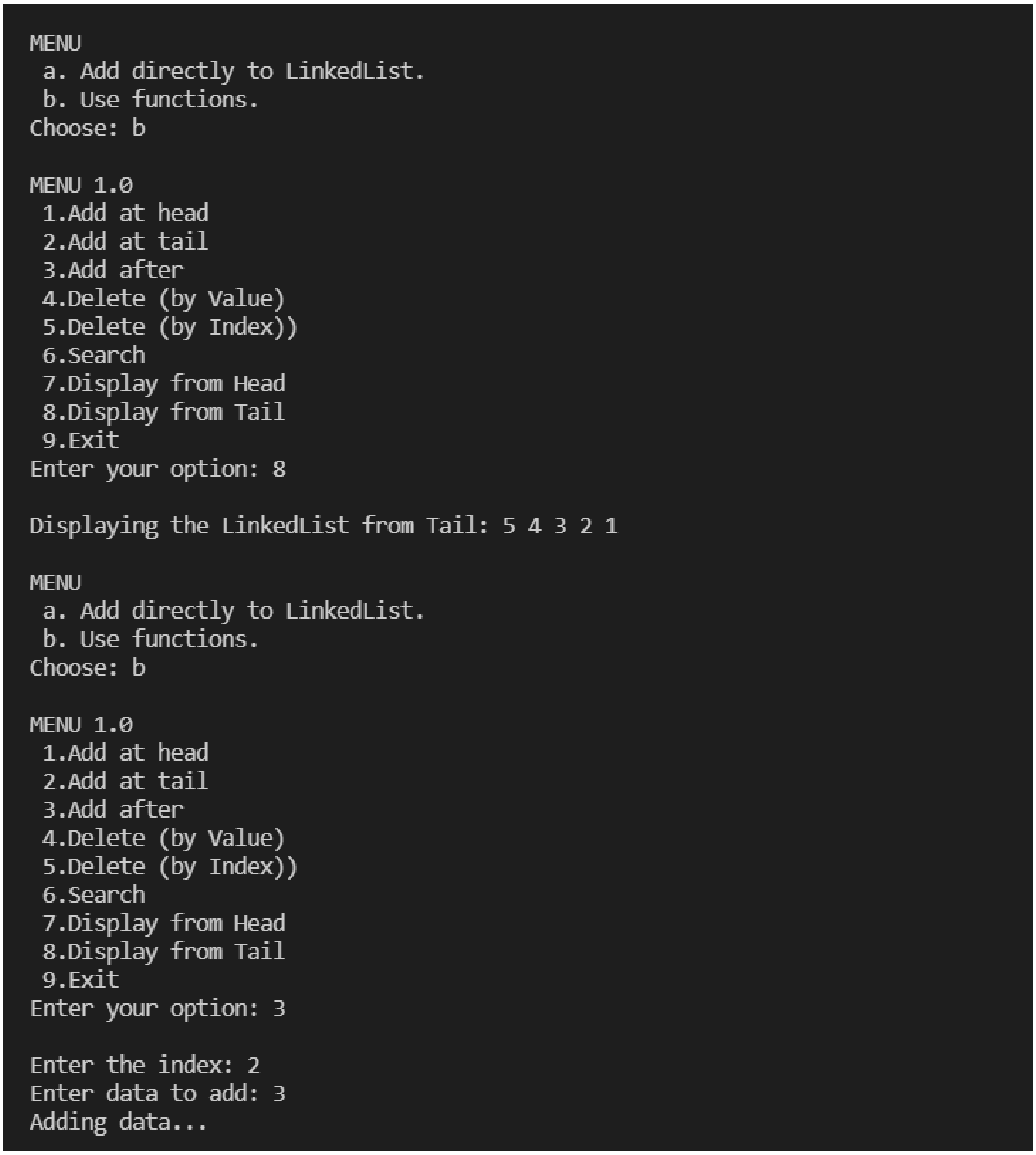


```
                    break;

                case 8:
                    cout<<"Displaying the LinkedList from Tail: ";
                    printListTail(h);
                    cout<<"\n";
                    break;

                case 9:
                    break;

                default:
                     break;

            }

        default:
            break;
        }
    }

    return 0;
}
```

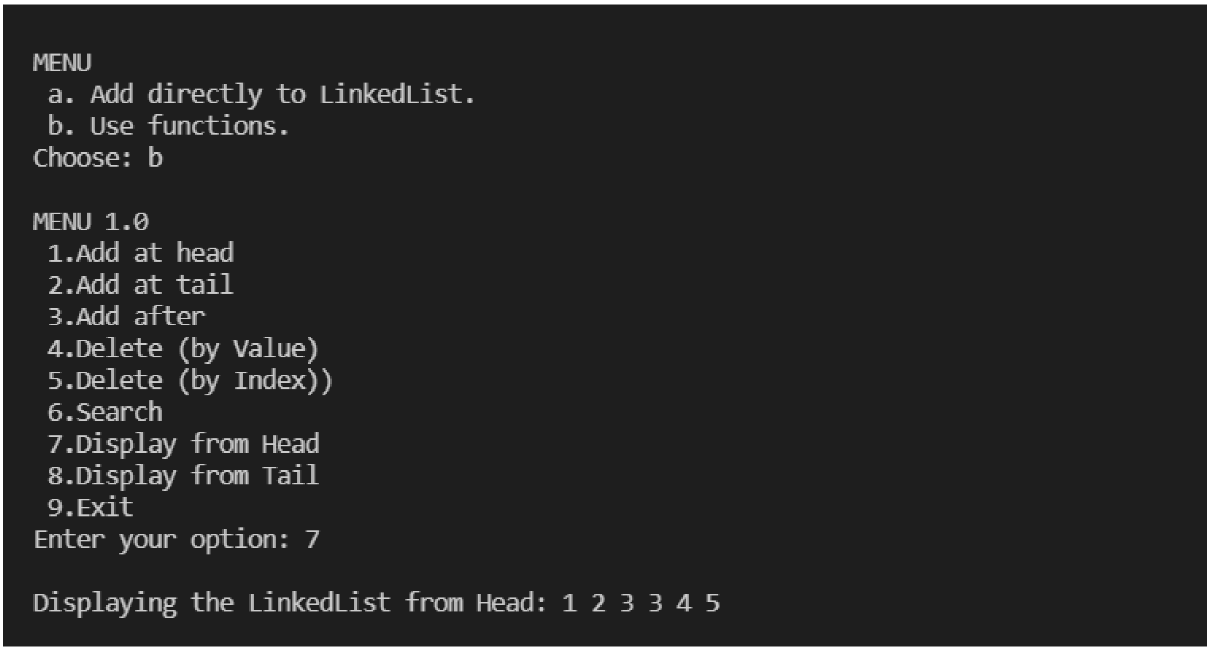
OUTPUT:



```
MENU
 a. Add directly to LinkedList.
 b. Use functions.
Choose: a
Enter the data to add in LinkedList (enter '-1' if you want stop): 1 2 3 4 5
-1

MENU
 a. Add directly to LinkedList.
 b. Use functions.
Choose: b

MENU 1.0
 1.Add at head
 2.Add at tail
 3.Add after
 4.Delete (by Value)
 5.Delete (by Index))
 6.Search
 7.Display from Head
 8.Display from Tail
 9.Exit
Enter your option: 7

Displaying the LinkedList from Head: 1 2 3 4 5
```



```
MENU
 a. Add directly to LinkedList.
 b. Use functions.
Choose: b

MENU 1.0
 1.Add at head
 2.Add at tail
 3.Add after
 4.Delete (by Value)
 5.Delete (by Index))
 6.Search
 7.Display from Head
 8.Display from Tail
 9.Exit
Enter your option: 8

Displaying the LinkedList from Tail: 5 4 3 2 1

MENU
 a. Add directly to LinkedList.
 b. Use functions.
Choose: b

MENU 1.0
 1.Add at head
 2.Add at tail
 3.Add after
 4.Delete (by Value)
 5.Delete (by Index))
 6.Search
 7.Display from Head
 8.Display from Tail
 9.Exit
Enter your option: 3

Enter the index: 2
Enter data to add: 3
Adding data...
```



```
MENU
 a. Add directly to LinkedList.
 b. Use functions.
Choose: b

MENU 1.0
 1.Add at head
 2.Add at tail
 3.Add after
 4.Delete (by Value)
 5.Delete (by Index))
 6.Search
 7.Display from Head
 8.Display from Tail
 9.Exit
Enter your option: 7

Displaying the LinkedList from Head: 1 2 3 3 4 5
```

ALGORITHM:

Algorithm to create Doubly Linked list

```
Begin:
alloc (head)
If (head == NULL) then
write ('Unable to allocate memory')
End if
Else then
read (data)
head.data ← data;
head.prev ← NULL;
head.next ← NULL;
last ← head;
write ('List created successfully')
End else
End
```

Algorithm to traverse Doubly Linked list

```
Input : head {Pointer to the first node of the list}
Begin:
    If (head == NULL) then
        write ('List is empty')
    End if
    Else then
        temp ← head;
        While (temp != NULL) do
            write ('Data = ', temp.data)
            temp ← temp.next;
        End while
    End else
End
```

Algorithm to insert a node at the beginning:

Input : head {A pointer pointing to the first node of the list}

```
Begin:
    alloc (newNode)
    If (newNode == NULL) then
        write ('Unable to allocate memory')
    End if
    Else then
        read (data)
        newNode.data ← data;
        newNode.prev ← NULL;
        newNode.next ← head;
        head.prev ← newNode;
        head ← newNode;
        write('Node added successfully at the beginning of List')
    End else
End
```

Algorithm to insert a node at the end of Doubly linked list

Input : last {Pointer to the last node of doubly linked list}

```
Begin:
```

```
alloc (newNode)
If (newNode == NULL) then
    write ('Unable to allocate memory')
End if
Else then
    read (data)
    newNode.data ← data;
    newNode.next ← NULL;
    newNode.prev ← last;
    last.next ← newNode;
    last ← newNode;
    write ('Node added successfully at the end of List')
End else
End
```

Algorithm to insert node at any position of doubly linked list

```
Input : head {Pointer to the first node of doubly linked list}
: last {Pointer to the last node of doubly linked list}
: N {Position where node is to be inserted}
Begin:
    temp ← head
    For i←1 to N-1 do
        If (temp == NULL) then
            break
        End if
        temp ← temp.next;
    End for
    If (N == 1) then
        insertAtBeginning()
    End if
    Else If (temp == last) then
        insertAtEnd()
```

```
End if
Else If (temp != NULL) then
alloc (newNode)
read (data)
newNode.data ← data;
newNode.next ← temp.next
newNode.prev ← temp
If (temp.next != NULL) then
temp.next.prev ← newNode;
End if
temp.next ← newNode;
write('Node added successfully')
End if
End
```

Algorithm to delete node from beginning

```
Input: head {Pointer to first node of the linked list}
Begin:
If (head == NULL) then
write ('Can't delete from an empty list')
End if
Else then
toDelete ← head;
head ← head.next;
head.prev ← NULL;
unalloc (toDelete)
write ('Successfully deleted first node from the list')
End if
End
```

Algorithm to delete node from end

```
Input: last {Pointer to last node of the linked list}
Begin:
```

```
If (last == NULL) then
    write ('Can't delete from an empty list')
End if
Else then
    toDelete ← last;
    last ← last.prev;
    last.next ← NULL;
    unalloc (toDelete)
    write ('Successfully deleted last node from the list')
End if
End
```

Algorithm to delete node from any position

Input : head {Pointer to the first node of the list}

last {Pointer to the last node of the list}

N {Position to be deleted from list}

```
Begin:
    current ← head;
    For i ← 1 to N and current != NULL do
        current ← current.next;
    End for
    If (N == 1) then
        deleteFromBeginning()
    End if
    Else if (current == last) then
        deleteFromEnd()
    End if
    Else if (current != NULL) then
        current.prev.next ← current.next
        If (current.next != NULL) then
            current.next.prev ← current.prev;
        End if
```

```
unalloc (current)
write ('Node deleted successfully from ', N, ' position')
End if
Else then
write ('Invalid position')
End if
End
```

Algorithm to insert a node at the beginning:

```
Input : head {A pointer pointing to the first node of the list}
Data {Data to search}
Begin:
alloc (newNode)
while(newNode != NULL)
{
if(newNode->data == Data) then
return 1;
newNode = newNode->next;
}
return 0;
End
```