Q1. Write a menu driven program to perform the following operations in a single linked list by using suitable user defined functions for each case.

Traversal of the list.

Check if the list is empty.

Insert a node at the certain position (at beginning/end/any position).

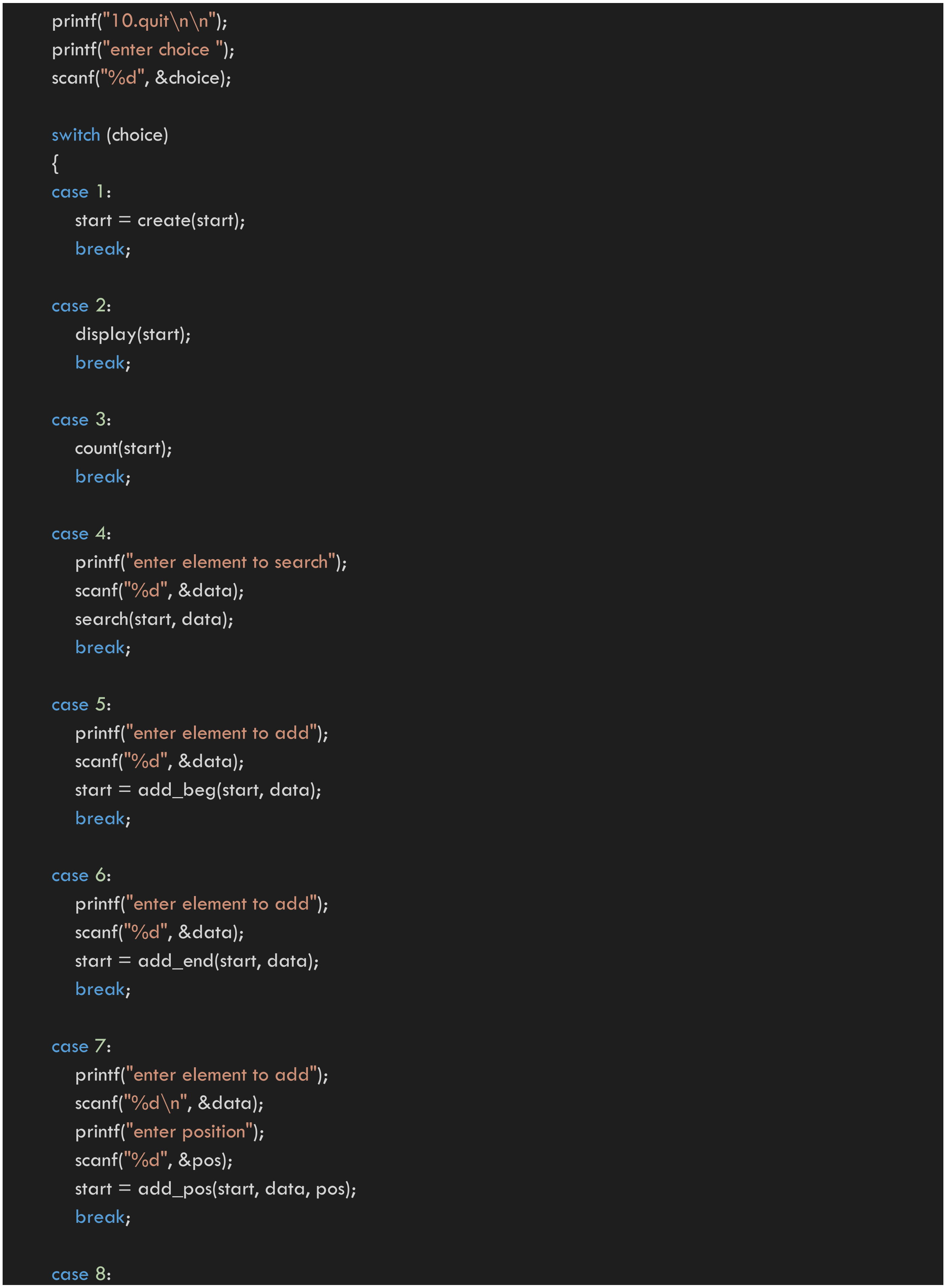
Delete a node at the certain position (at beginning/end/any position).

Delete a node for the given key.

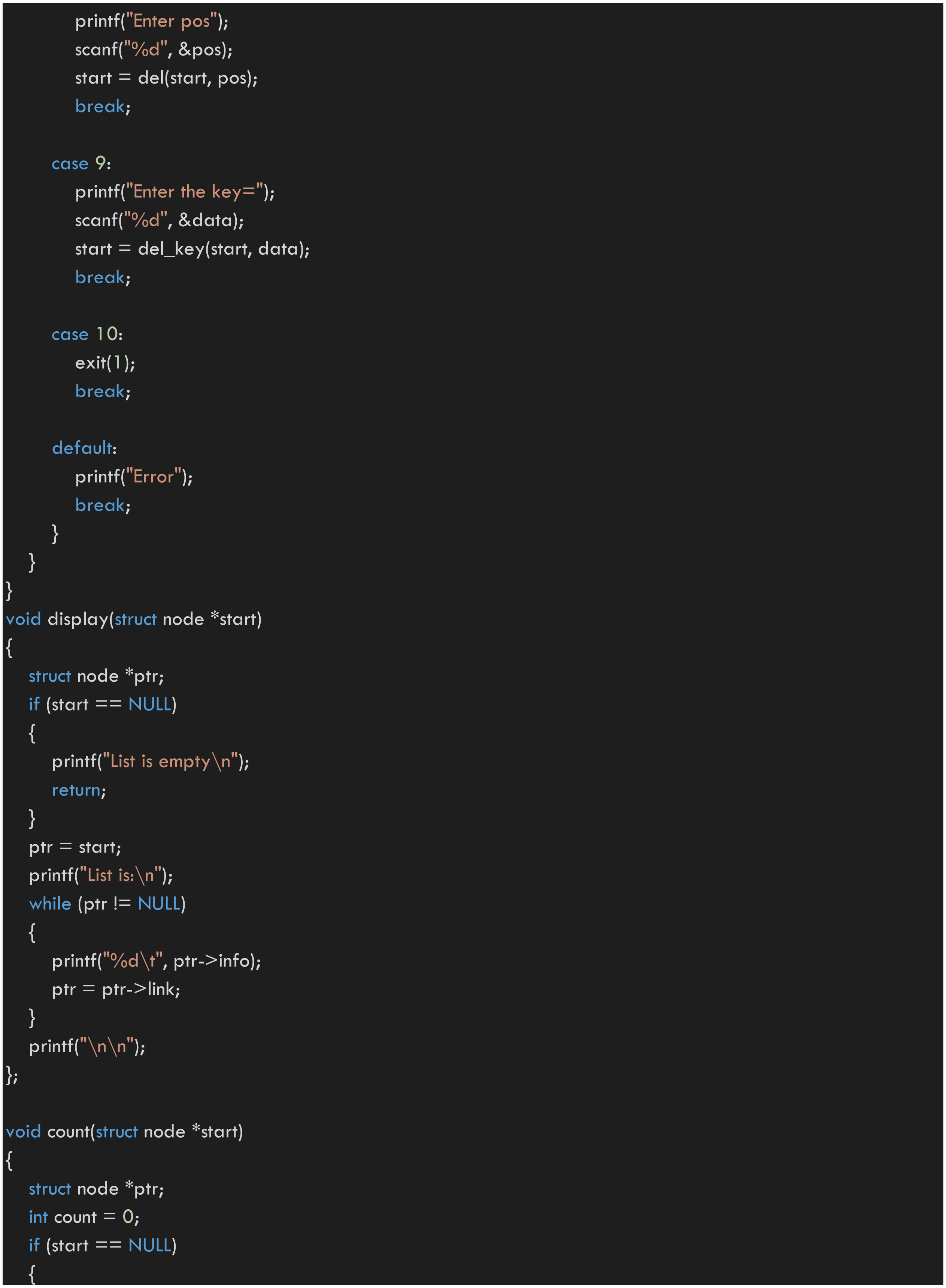
Count the total number of nodes.

Search for an element in the linked list.

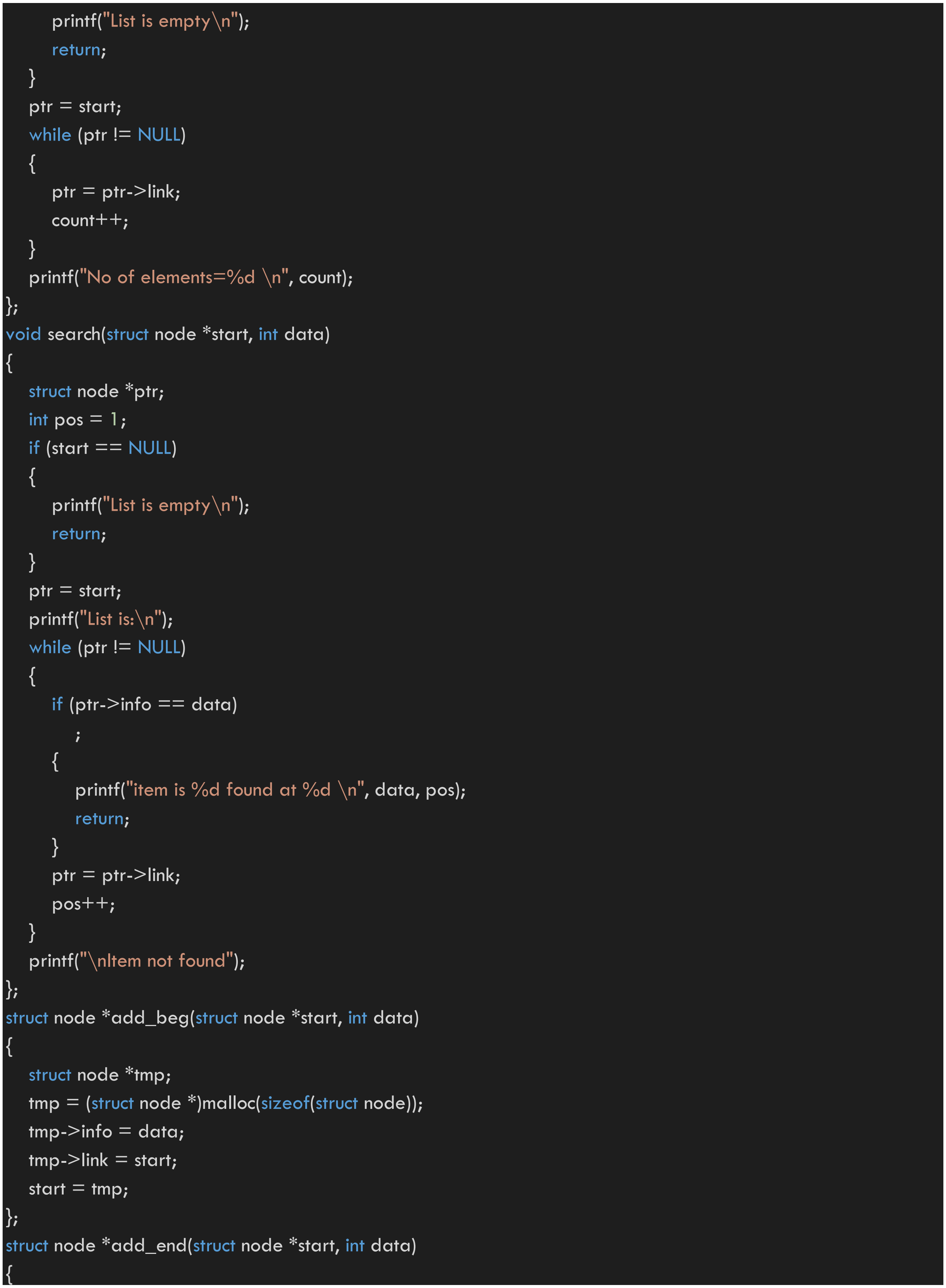
```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int info;
    struct node *link;
};
struct node *create(struct node *start);
void display(struct node *start);
void count(struct node *start);
void search(struct node *start, int data);
struct node *add_beg(struct node *start, int data);
struct node *add_end(struct node *start, int data);
struct node *add_pos(struct node *start, int data, int pos);
struct node *del(struct node *start, int pos);
struct node *del_key(struct node *start, int item);

int main(void)
{
    struct node *start = NULL;
    int choice, pos, data, item;
    while (1)
    {
        printf("1.create list\n");
        printf("2.Traverse\n");
        printf("3.count\n");
        printf("4.search\n");
        printf("5.Add in the beginning\n");
        printf("6.Add in the end\n");
        printf("7.Add at any pos\n");
        printf("8.Delete at any pos\n");
        printf("9.Delete for a given key\n");
```



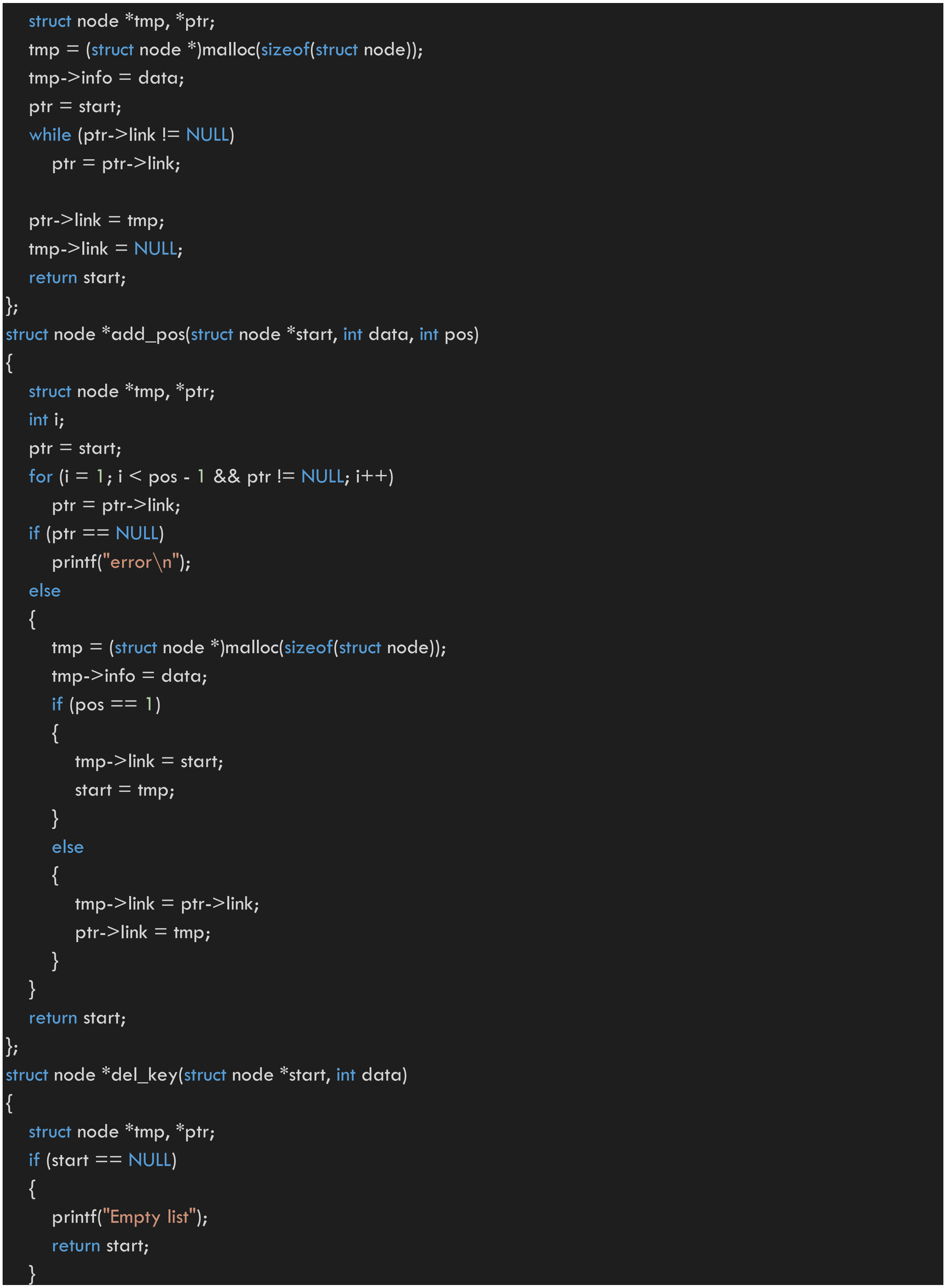
```
    printf("10.quit\n\n");
    printf("enter choice ");
    scanf("%d", &choice);

    switch (choice)
    {
    case 1:
        start = create(start);
        break;

    case 2:
        display(start);
        break;

    case 3:
        count(start);
        break;

    case 4:
        printf("enter element to search");
        scanf("%d", &data);
        search(start, data);
        break;

    case 5:
        printf("enter element to add");
        scanf("%d", &data);
        start = add_beg(start, data);
        break;

    case 6:
        printf("enter element to add");
        scanf("%d", &data);
        start = add_end(start, data);
        break;

    case 7:
        printf("enter element to add");
        scanf("%d\n", &data);
        printf("enter position");
        scanf("%d", &pos);
        start = add_pos(start, data, pos);
        break;

    case 8:
```



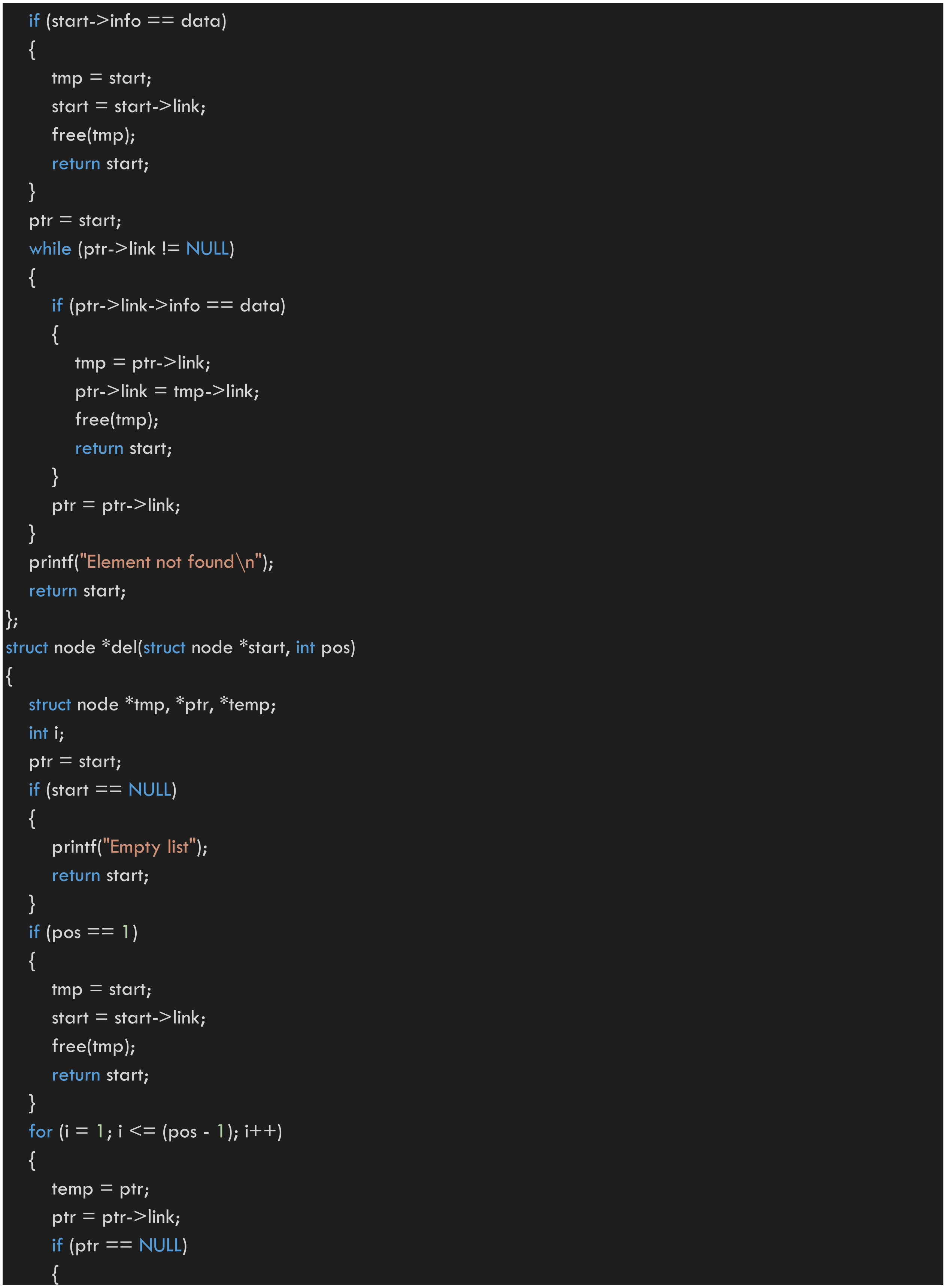
```
            printf("Enter pos");
            scanf("%d", &pos);
            start = del(start, pos);
            break;

        case 9:
            printf("Enter the key=");
            scanf("%d", &data);
            start = del_key(start, data);
            break;

        case 10:
            exit(1);
            break;

        default:
            printf("Error");
            break;
        }
    }
}
void display(struct node *start)
{
    struct node *ptr;
    if (start == NULL)
    {
        printf("List is empty\n");
        return;
    }
    ptr = start;
    printf("List is:\n");
    while (ptr != NULL)
    {
        printf("%d\t", ptr->info);
        ptr = ptr->link;
    }
    printf("\n\n");
};

void count(struct node *start)
{
    struct node *ptr;
    int count = 0;
    if (start == NULL)
    {
```



```
        printf("List is empty\n");
        return;
    }
    ptr = start;
    while (ptr != NULL)
    {
        ptr = ptr->link;
        count++;
    }
    printf("No of elements=%d \n", count);
};
void search(struct node *start, int data)
{
    struct node *ptr;
    int pos = 1;
    if (start == NULL)
    {
        printf("List is empty\n");
        return;
    }
    ptr = start;
    printf("List is:\n");
    while (ptr != NULL)
    {
        if (ptr->info == data)
            ;
        {
            printf("item is %d found at %d \n", data, pos);
            return;
        }
        ptr = ptr->link;
        pos++;
    }
    printf("\nItem not found");
};
struct node *add_beg(struct node *start, int data)
{
    struct node *tmp;
    tmp = (struct node *)malloc(sizeof(struct node));
    tmp->info = data;
    tmp->link = start;
    start = tmp;
};
struct node *add_end(struct node *start, int data)
{
```



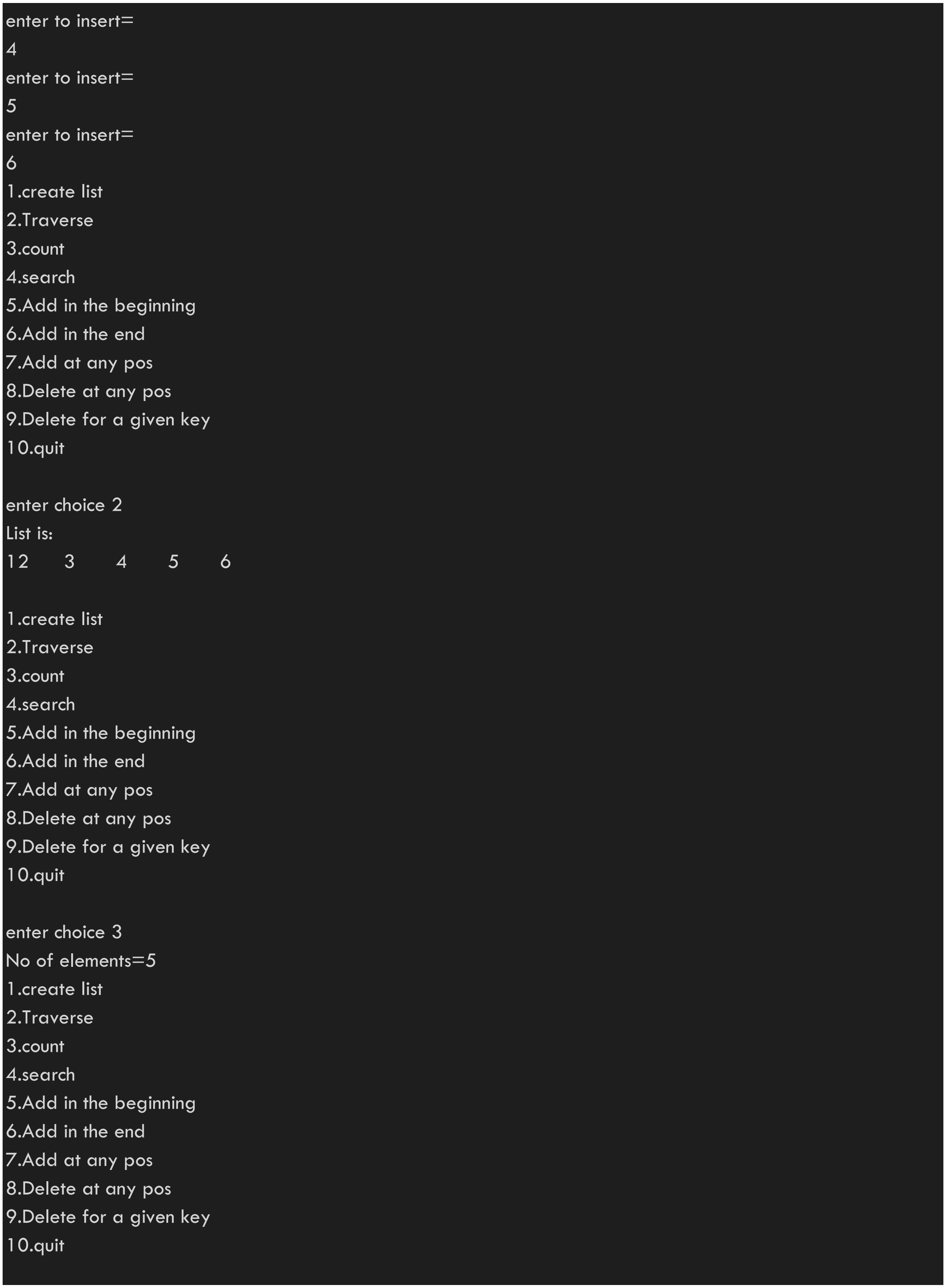
```
    struct node *tmp, *ptr;
    tmp = (struct node *)malloc(sizeof(struct node));
    tmp->info = data;
    ptr = start;
    while (ptr->link != NULL)
        ptr = ptr->link;

    ptr->link = tmp;
    tmp->link = NULL;
    return start;
};
struct node *add_pos(struct node *start, int data, int pos)
{
    struct node *tmp, *ptr;
    int i;
    ptr = start;
    for (i = 1; i < pos - 1 && ptr != NULL; i++)
        ptr = ptr->link;
    if (ptr == NULL)
        printf("error\n");
    else
    {
        tmp = (struct node *)malloc(sizeof(struct node));
        tmp->info = data;
        if (pos == 1)
        {
            tmp->link = start;
            start = tmp;
        }
        else
        {
            tmp->link = ptr->link;
            ptr->link = tmp;
        }
    }
    return start;
};
struct node *del_key(struct node *start, int data)
{
    struct node *tmp, *ptr;
    if (start == NULL)
    {
        printf("Empty list");
        return start;
    }
```



```
    if (start->info == data)
    {
        tmp = start;
        start = start->link;
        free(tmp);
        return start;
    }
    ptr = start;
    while (ptr->link != NULL)
    {
        if (ptr->link->info == data)
        {
            tmp = ptr->link;
            ptr->link = tmp->link;
            free(tmp);
            return start;
        }
        ptr = ptr->link;
    }
    printf("Element not found\n");
    return start;
};
struct node *del(struct node *start, int pos)
{
    struct node *tmp, *ptr, *temp;
    int i;
    ptr = start;
    if (start == NULL)
    {
        printf("Empty list");
        return start;
    }
    if (pos == 1)
    {
        tmp = start;
        start = start->link;
        free(tmp);
        return start;
    }
    for (i = 1; i <= (pos - 1); i++)
    {
        temp = ptr;
        ptr = ptr->link;
        if (ptr == NULL)
        {
            printf("Invalid pos\n");
        }
        temp->link = ptr->link;
        free(ptr);
        return start;
    }
}
struct node *create(struct node *start)
{
    int i, n, data;
    printf("Enter no of nodes:");
    scanf("%d", &n);
    start = NULL;
    if (n == 0)
        return start;
    printf("Enter element to enter:");
    scanf("%d", &data);
    start = add_beg(start, data);
    for (i = 2; i <= n; i++)
    {
        printf("enter to insert=\n");
        scanf("%d", &data);
        start = add_end(start, data);
    }
    return start;
}
```

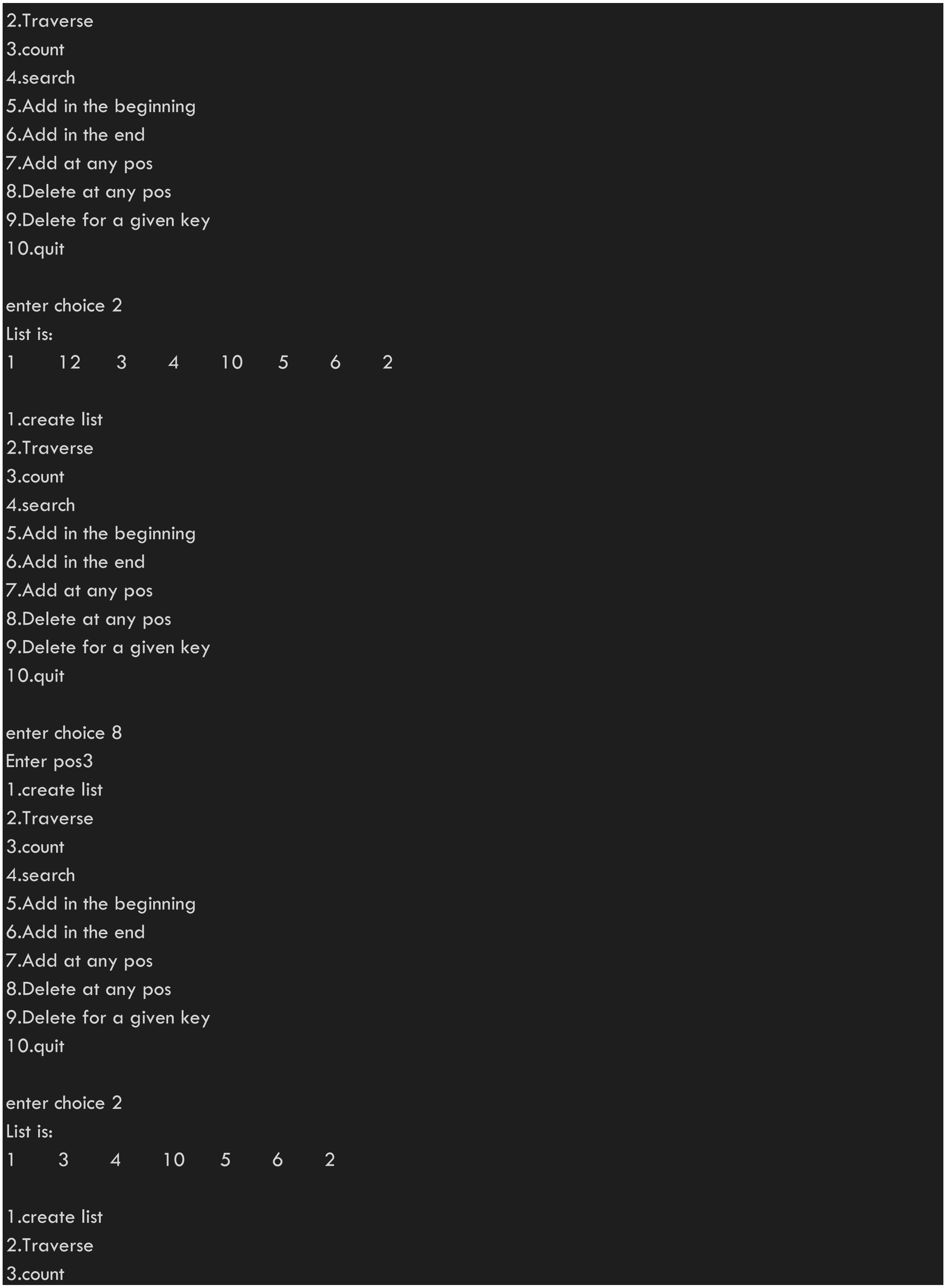
OUTPUT:-

```
1.create list
2.Traverse
3.count
4.search
5.Add in the beginning
6.Add in the end
7.Add at any pos
8.Delete at any pos
9.Delete for a given key
10.quit

enter choice 1
Enter no of nodes:5
Enter element to enter:12
enter to insert=
3
```



```
enter to insert=
4
enter to insert=
5
enter to insert=
6
1.create list
2.Traverse
3.count
4.search
5.Add in the beginning
6.Add in the end
7.Add at any pos
8.Delete at any pos
9.Delete for a given key
10.quit

enter choice 2
List is:
12      3       4       5       6

1.create list
2.Traverse
3.count
4.search
5.Add in the beginning
6.Add in the end
7.Add at any pos
8.Delete at any pos
9.Delete for a given key
10.quit

enter choice 3
No of elements=5
1.create list
2.Traverse
3.count
4.search
5.Add in the beginning
6.Add in the end
7.Add at any pos
8.Delete at any pos
9.Delete for a given key
10.quit
```

```
enter choice 4
enter element to search 3
List is:
item is 3 found at 1
1.create list
2.Traverse
3.count
4.search
5.Add in the beginning
6.Add in the end
7.Add at any pos
8.Delete at any pos
9.Delete for a given key
10.quit

enter choice 5
enter element to add1
1.create list
2.Traverse
3.count
4.search
5.Add in the beginning
6.Add in the end
7.Add at any pos
8.Delete at any pos
9.Delete for a given key
10.quit

enter choice 6
enter element to add2
1.create list
2.Traverse
3.count
4.search
5.Add in the beginning
6.Add in the end
7.Add at any pos
8.Delete at any pos
9.Delete for a given key
10.quit

enter choice 7
enter element to add10
5
enter position1.create list
```



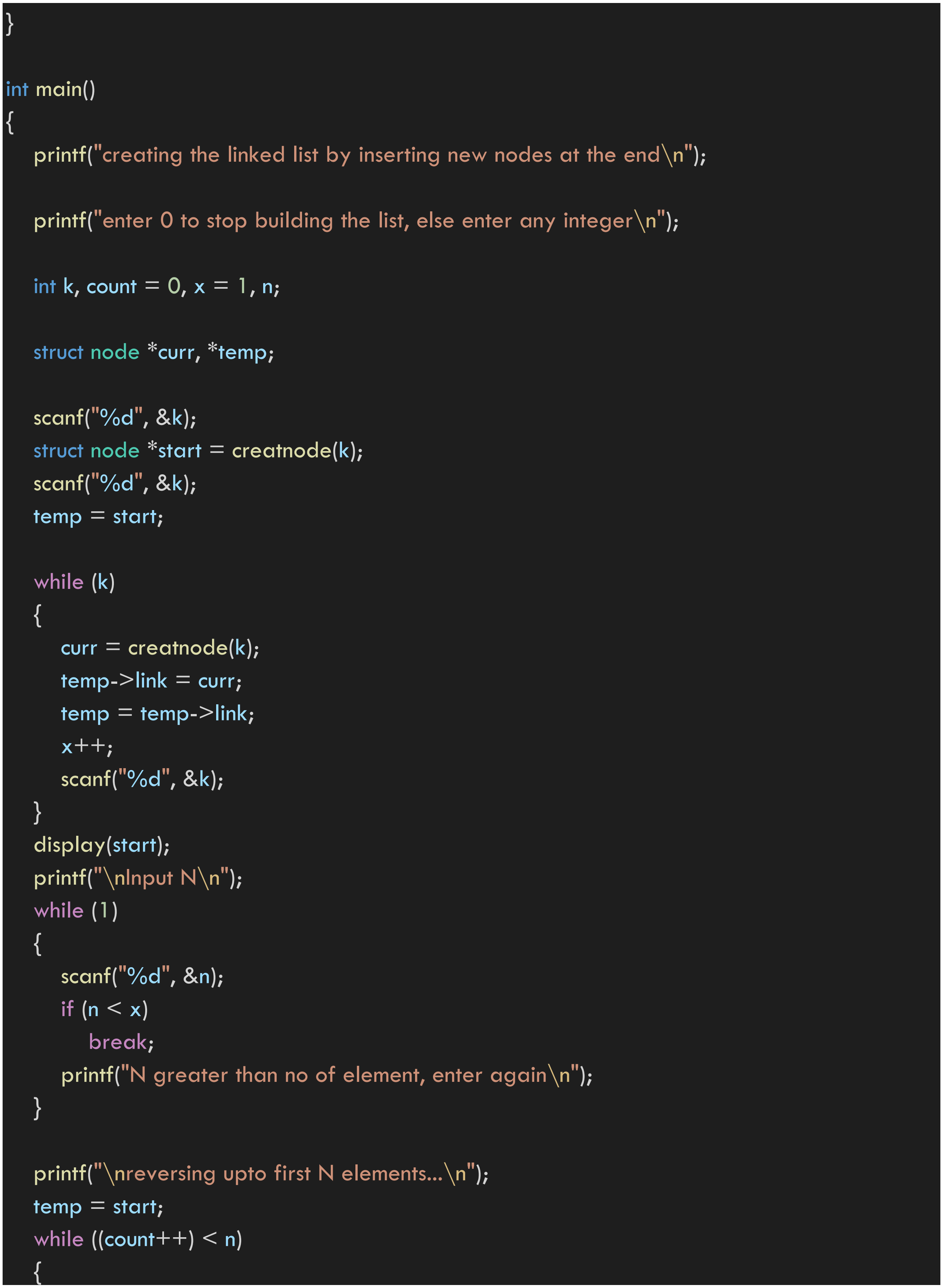
```
2.Traverse
3.count
4.search
5.Add in the beginning
6.Add in the end
7.Add at any pos
8.Delete at any pos
9.Delete for a given key
10.quit

enter choice 2
List is:
1       12      3       4       10      5       6       2

1.create list
2.Traverse
3.count
4.search
5.Add in the beginning
6.Add in the end
7.Add at any pos
8.Delete at any pos
9.Delete for a given key
10.quit

enter choice 8
Enter pos3
1.create list
2.Traverse
3.count
4.search
5.Add in the beginning
6.Add in the end
7.Add at any pos
8.Delete at any pos
9.Delete for a given key
10.quit

enter choice 2
List is:
1       3       4       10      5       6       2

1.create list
2.Traverse
3.count
```

```
4.search
5.Add in the beginning
6.Add in the end
7.Add at any pos
8.Delete at any pos
9.Delete for a given key
10.quit

enter choice 9
Enter the key=3
1.create list
2.Traverse
3.count
4.search
5.Add in the beginning
6.Add in the end
7.Add at any pos
8.Delete at any pos
9.Delete for a given key
10.quit

enter choice 2
List is:
1       4       10      5       6       2

1.create list
2.Traverse
3.count
4.search
5.Add in the beginning
6.Add in the end
7.Add at any pos
8.Delete at any pos
9.Delete for a given key
10.quit

enter choice 10
```

Q2. WAP to reverse the first m elements of a linked list of n nodes.

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


struct node
```

```
{
    int data;
    struct node *link;
};

void display(struct node *start)
{
    struct node *ptr = start;

    printf("traversing the list...\n");
    while (ptr != NULL)
    {
        printf("%d ", ptr->data);
        ptr = ptr->link;
    }
}

struct node *reverse_N(struct node *start, struct node *temp, int n)
{
    struct node *link = NULL, *cur = start, *prev = temp;
    int count = 0;
    while (cur != NULL && (count++) < n)
    {
        link = cur->link;
        cur->link = prev;
        prev = cur;
        cur = link;
    }

    start = prev;
    return start;
}

struct node *creatnode(int d)
{
    struct node *temp = malloc(sizeof(struct node));
    temp->data = d;
    temp->link = NULL;
    return temp;
```



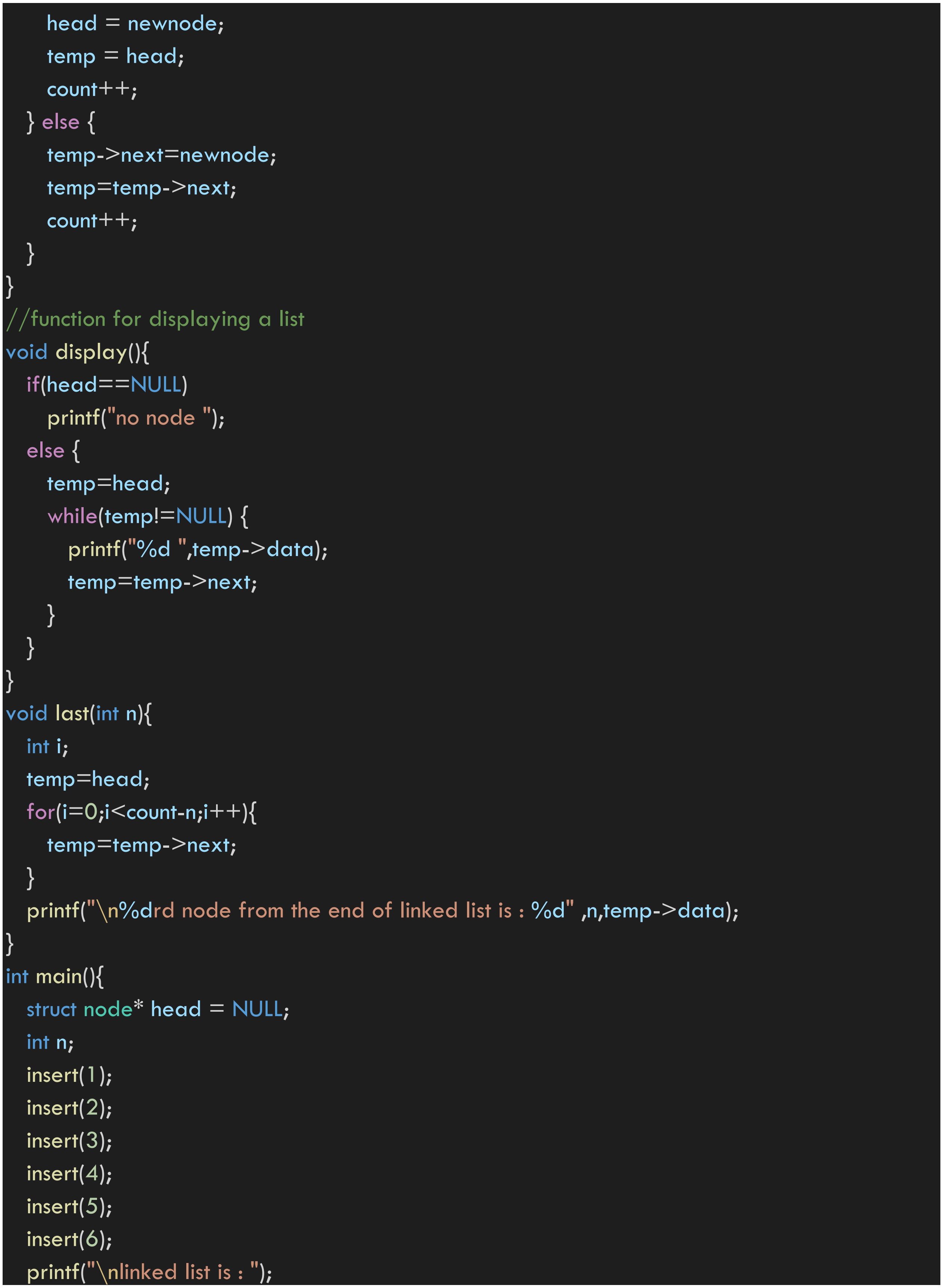
```
}

int main()
{
    printf("creating the linked list by inserting new nodes at the end\n");

    printf("enter 0 to stop building the list, else enter any integer\n");

    int k, count = 0, x = 1, n;

    struct node *curr, *temp;

    scanf("%d", &k);
    struct node *start = creatnode(k);
    scanf("%d", &k);
    temp = start;

    while (k)
    {
        curr = creatnode(k);
        temp->link = curr;
        temp = temp->link;
        x++;
        scanf("%d", &k);
    }
    display(start);
    printf("\nInput N\n");
    while (1)
    {
        scanf("%d", &n);
        if (n < x)
            break;
        printf("N greater than no of element, enter again\n");
    }

    printf("\nreversing upto first N elements...\n");
    temp = start;
    while ((count++) < n)
    {
        temp = temp->link;
    }

    start = reverse_N(start, temp, n);
    display(start);

    return 0;
}
```

OUTPUT:-

```
creating the linked list by inserting new nodes at the end
enter 0 to stop building the list, else enter any integer
9 8 7 6 5 4 3 2 1 10 0
traversing the list...
9 8 7 6 5 4 3 2 1 10
Input N
3

reversing upto first N elements...
traversing the list...
7 8 9 6 5 4 3 2 1 10
```

Q3. WAP to print m th node from the last of a linked list of n nodes.

```
#include<stdio.h>
#include<stdlib.h>
//structure of a node
struct node{
   int data;
   struct node *next;
}*head,*temp;
int count=0;

void insert(int val){
   struct node* newnode = (struct node*)malloc(sizeof(struct node));
   newnode->data = val;
   newnode->next = NULL;
   if(head == NULL){
```



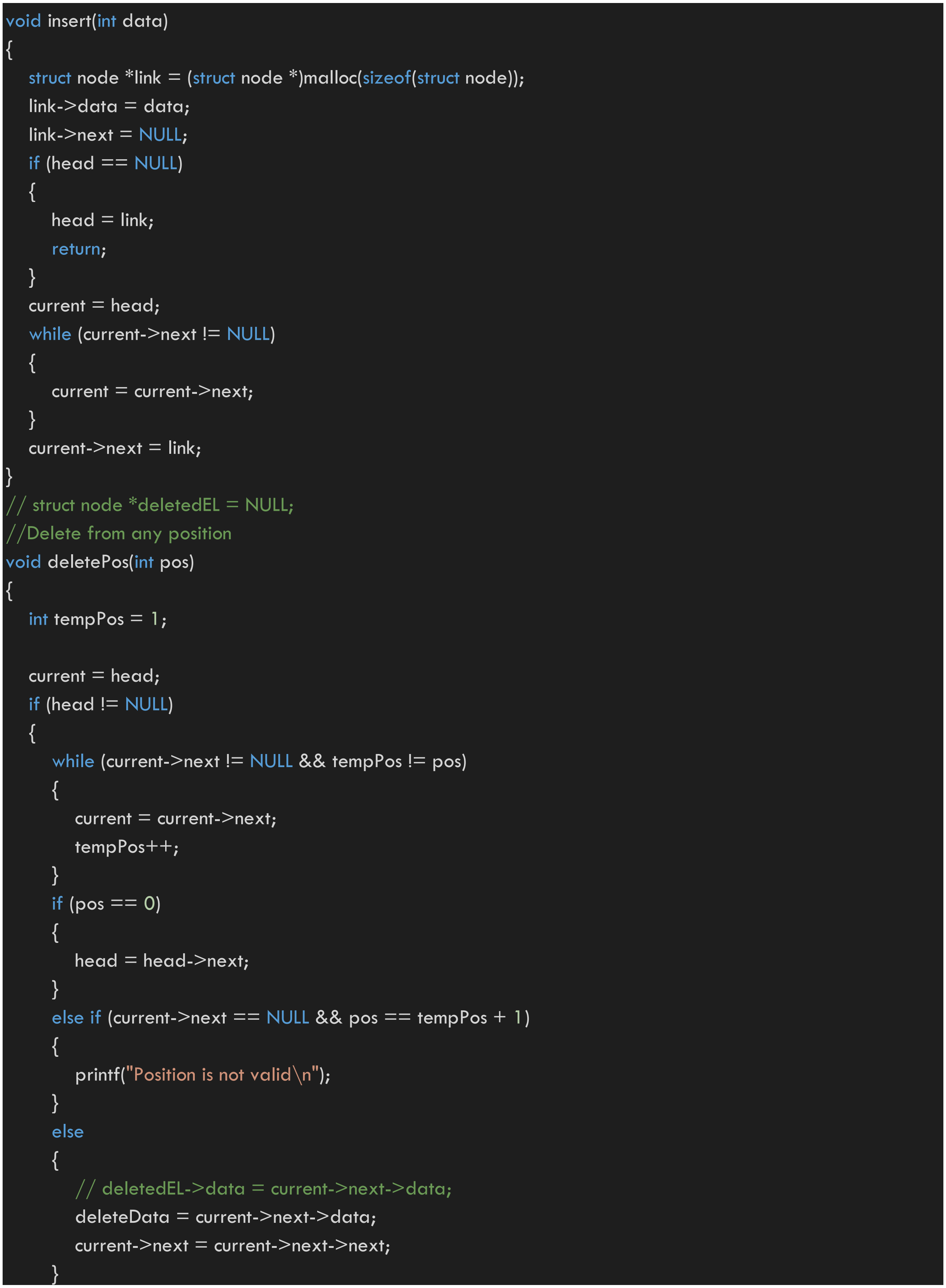
```
    head = newnode;
    temp = head;
    count++;
  } else {
    temp->next=newnode;
    temp=temp->next;
    count++;
  }
}
//function for displaying a list
void display(){
  if(head==NULL)
    printf("no node ");
  else {
    temp=head;
    while(temp!=NULL) {
      printf("%d ",temp->data);
      temp=temp->next;
    }
  }
}
void last(int n){
  int i;
  temp=head;
  for(i=0;i<count-n;i++){
    temp=temp->next;
  }
  printf("\n%drd node from the end of linked list is : %d" ,n,temp->data);
}
int main(){
  struct node* head = NULL;
  int n;
  insert(1);
  insert(2);
  insert(3);
  insert(4);
  insert(5);
  insert(6);
  printf("\nlinked list is : ");
  display();
  printf("\nEnter the node you want to find from last\n");
  scanf("%d", &n);
  last(n);
  return 0;
}
```

OUTPUT:-

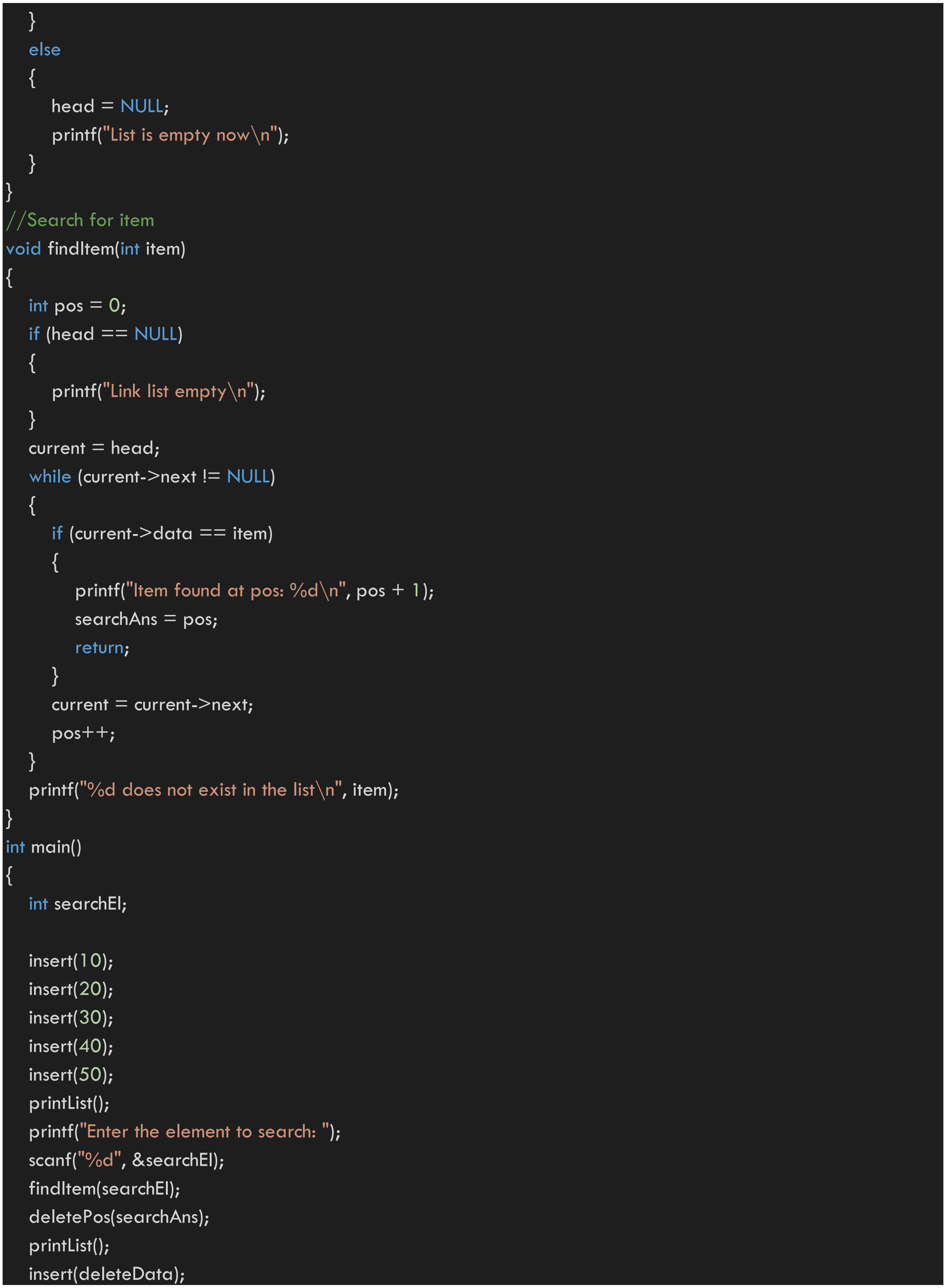
```
linked list is : 1 2 3 4 5 6
Enter the node you want to find from last
3

3rd node from the end of linked list is : 4
```

Q4. WAP to search an element in a simple linked list, if found delete that node and insert that node at beginning. Otherwise display an appropriate message.

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *next;
};
struct node *head = NULL;
struct node *current = NULL;
int searchAns = 0;
int deleteData = 0;
//Traversal of the list
void printList()
{
    struct node *ptr = head;
    printf("\n[head] => ");
    while (ptr != NULL)
    {
        printf(" %d =>", ptr->data);
        ptr = ptr->next;
    }

    printf(" [null]\n");
}
```



```
void insert(int data)
{
    struct node *link = (struct node *)malloc(sizeof(struct node));
    link->data = data;
    link->next = NULL;
    if (head == NULL)
    {
        head = link;
        return;
    }
    current = head;
    while (current->next != NULL)
    {
        current = current->next;
    }
    current->next = link;
}
// struct node *deletedEL = NULL;
//Delete from any position
void deletePos(int pos)
{
    int tempPos = 1;

    current = head;
    if (head != NULL)
    {
        while (current->next != NULL && tempPos != pos)
        {
            current = current->next;
            tempPos++;
        }
        if (pos == 0)
        {
            head = head->next;
        }
        else if (current->next == NULL && pos == tempPos + 1)
        {
            printf("Position is not valid\n");
        }
        else
        {
            // deletedEL->data = current->next->data;
            deleteData = current->next->data;
            current->next = current->next->next;
        }
```



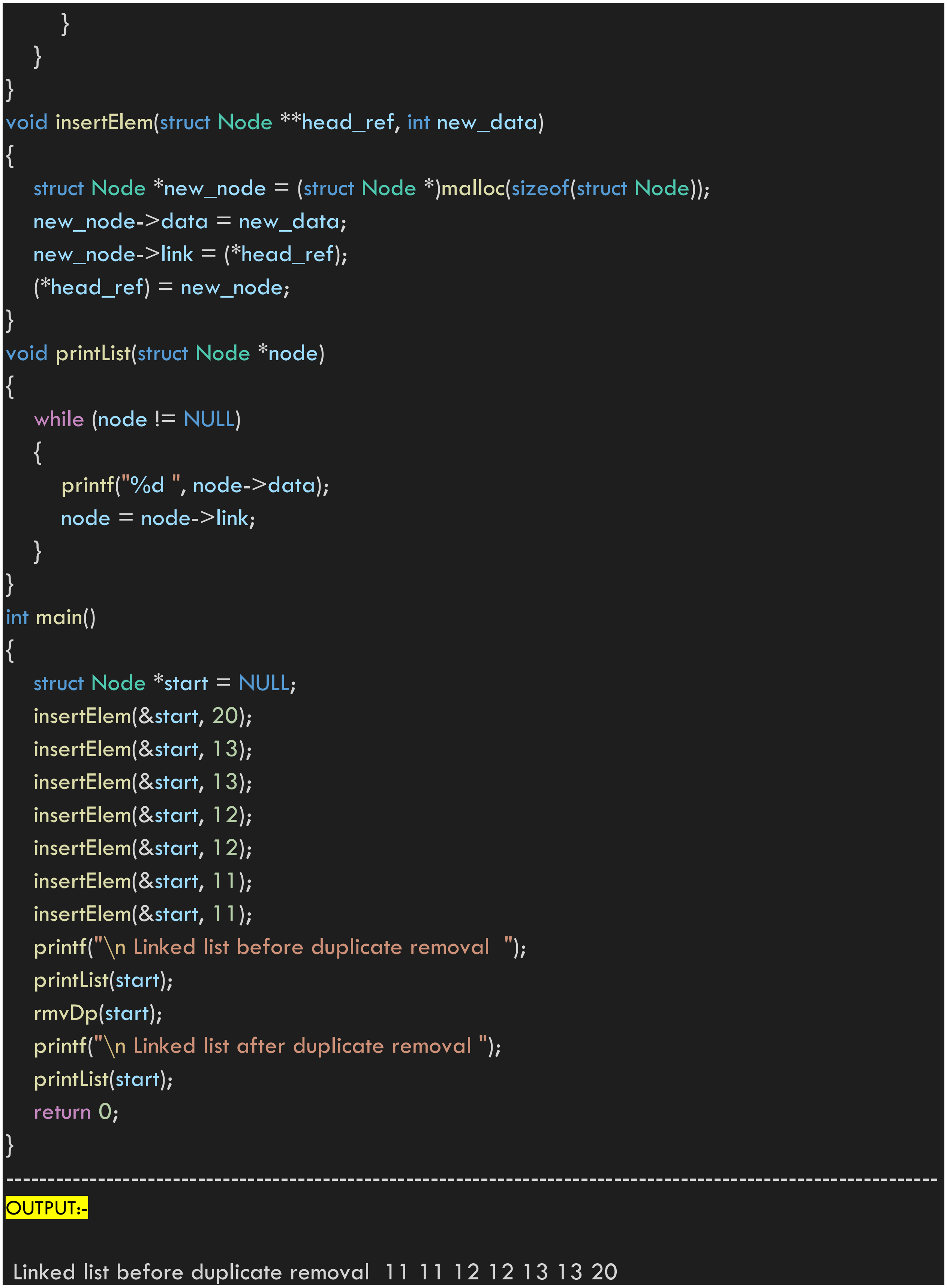
```
    }
    else
    {
        head = NULL;
        printf("List is empty now\n");
    }
}
//Search for item
void findItem(int item)
{
    int pos = 0;
    if (head == NULL)
    {
        printf("Link list empty\n");
    }
    current = head;
    while (current->next != NULL)
    {
        if (current->data == item)
        {
            printf("Item found at pos: %d\n", pos + 1);
            searchAns = pos;
            return;
        }
        current = current->next;
        pos++;
    }
    printf("%d does not exist in the list\n", item);
}
int main()
{
    int searchEl;

    insert(10);
    insert(20);
    insert(30);
    insert(40);
    insert(50);
    printList();
    printf("Enter the element to search: ");
    scanf("%d", &searchEl);
    findItem(searchEl);
    deletePos(searchAns);
    printList();
    insert(deleteData);
    printList();
}
```

OUTPUT:-

```
[head] =>  10 => 20 => 30 => 40 => 50 => [null]
Enter the element to search: 20
Item found at pos: 2

[head] =>  10 => 30 => 40 => 50 => [null]

[head] =>  10 => 30 => 40 => 50 => 20 => [null]
```

Q5. WAP to remove duplicates from a linked list of n nodes.

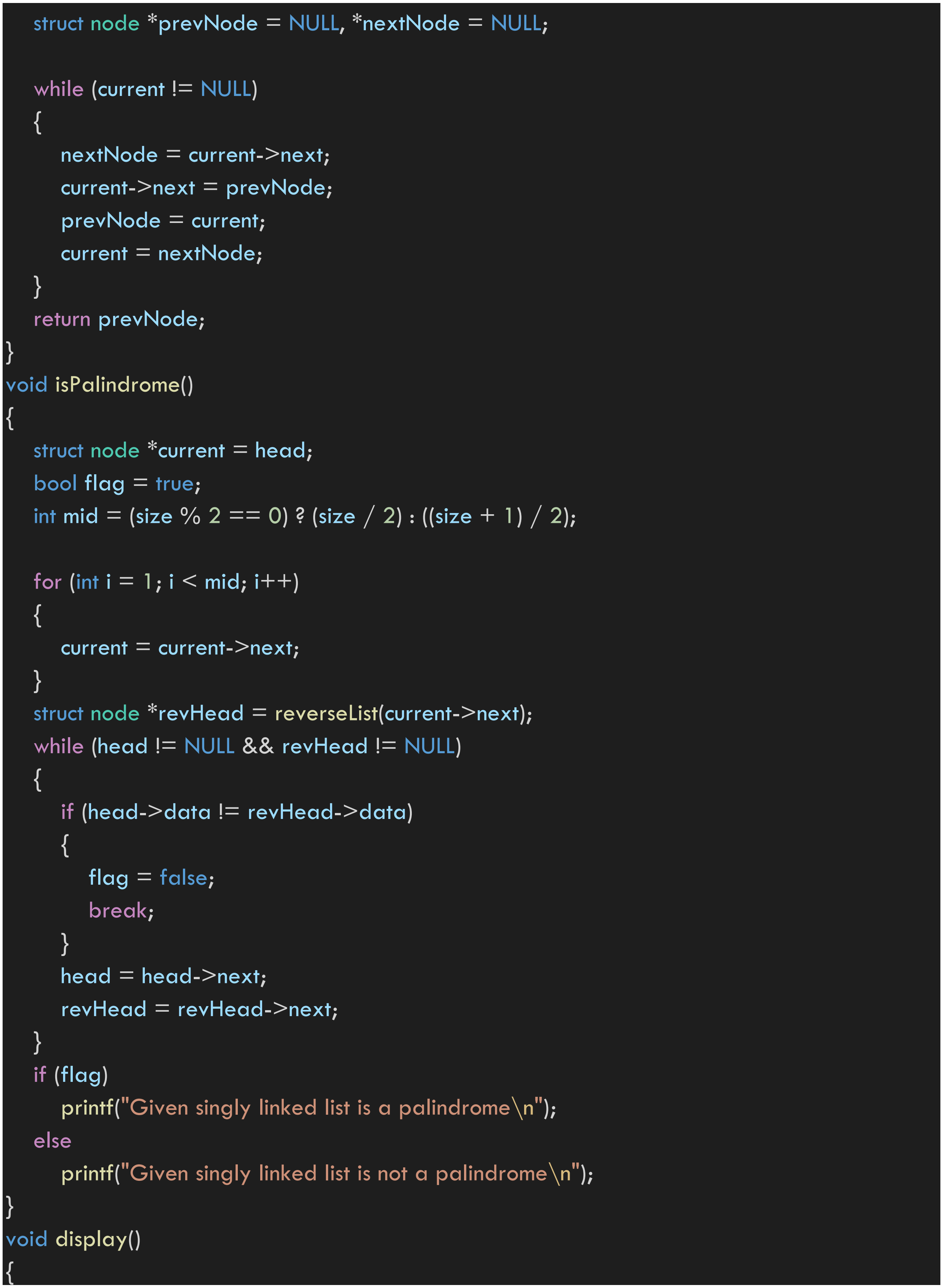
```
#include <stdio.h>
#include <stdlib.h>

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

void rmvDp(struct Node *start)
{
    struct Node *current = start;
    struct Node *next_next;
    if (current == NULL)
        return;
    while (current->link != NULL)
    {
        if (current->data == current->link->data)
        {
            next_next = current->link->link;
            free(current->link);
            current->link = next_next;
        }
        else
        {
            current = current->link;
```



```
        }
    }
}
void insertElem(struct Node **head_ref, int new_data)
{
    struct Node *new_node = (struct Node *)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->link = (*head_ref);
    (*head_ref) = new_node;
}
void printList(struct Node *node)
{
    while (node != NULL)
    {
        printf("%d ", node->data);
        node = node->link;
    }
}
int main()
{
    struct Node *start = NULL;
    insertElem(&start, 20);
    insertElem(&start, 13);
    insertElem(&start, 13);
    insertElem(&start, 12);
    insertElem(&start, 12);
    insertElem(&start, 11);
    insertElem(&start, 11);
    printf("\n Linked list before duplicate removal  ");
    printList(start);
    rmvDp(start);
    printf("\n Linked list after duplicate removal ");
    printList(start);
    return 0;
}
--------------------------------------------------------------------------------------------------------------------------
OUTPUT:-

 Linked list before duplicate removal  11 11 12 12 13 13 20
 Linked list after duplicate removal 11 12 13 20
```

Q6. WAP to check whether a singly linked list is a palindrome or not.

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

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

struct node *head, *tail = NULL;
int size = 0;

void addNode(int data)
{
    struct node *newNode = (struct node *)malloc(sizeof(struct node));
    newNode->data = data;
    newNode->next = NULL;

    if (head == NULL)
    {
        head = newNode;
        tail = newNode;
    }
    else
    {
        tail->next = newNode;
        tail = newNode;
    }
    size++;
}

struct node *reverseList(struct node *temp)
{
    struct node *current = temp;
```



```
    struct node *prevNode = NULL, *nextNode = NULL;

    while (current != NULL)
    {
        nextNode = current->next;
        current->next = prevNode;
        prevNode = current;
        current = nextNode;
    }
    return prevNode;
}
void isPalindrome()
{
    struct node *current = head;
    bool flag = true;
    int mid = (size % 2 == 0) ? (size / 2) : ((size + 1) / 2);

    for (int i = 1; i < mid; i++)
    {
        current = current->next;
    }
    struct node *revHead = reverseList(current->next);
    while (head != NULL && revHead != NULL)
    {
        if (head->data != revHead->data)
        {
            flag = false;
            break;
        }
        head = head->next;
        revHead = revHead->next;
    }
    if (flag)
        printf("Given singly linked list is a palindrome\n");
    else
        printf("Given singly linked list is not a palindrome\n");
}
void display()
{
    struct node *current = head;

    if (head == NULL)
    {
        printf("List is empty\n");
        return;
    }
    printf("Nodes of singly linked list: \n");
    while (current != NULL)
    {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}

int main()
{
    addNode(1);
    addNode(2);
    addNode(3);
    addNode(4);
    addNode(1);

    display();
    isPalindrome();
    return 0;
}
```

OUTPUT:-

```
Nodes of singly linked list:
1 2 3 4 1
Given singly linked list is not a palindrome
```
