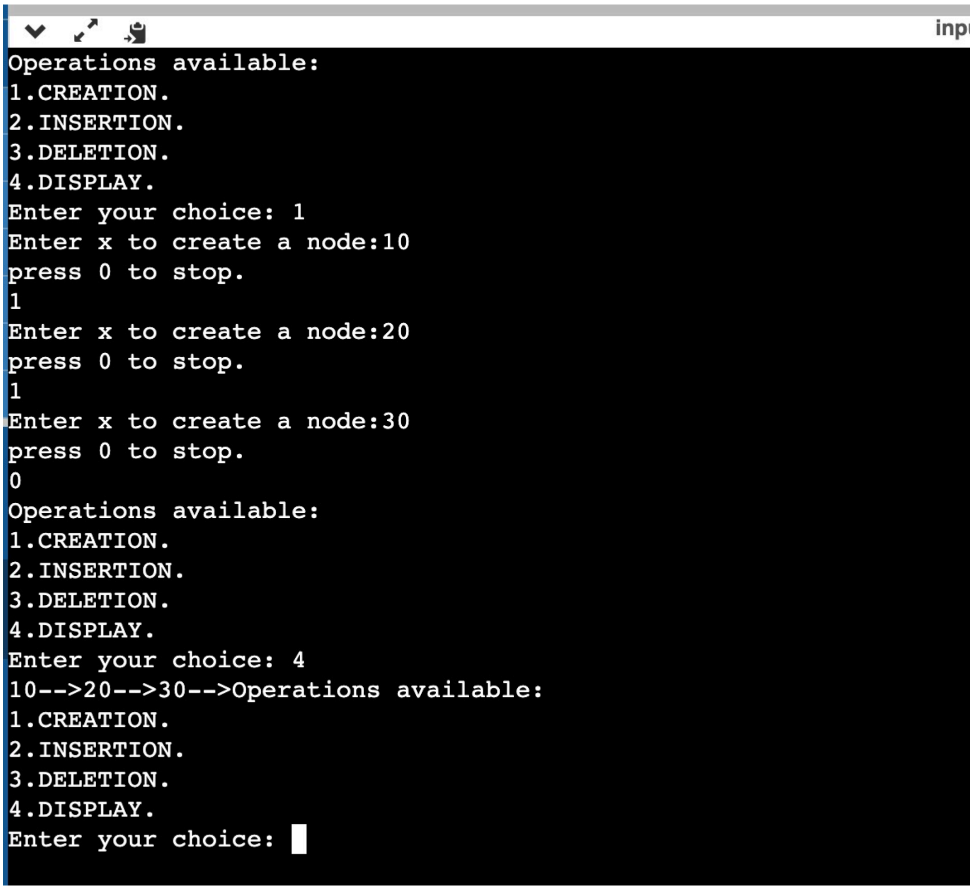
Name: Riya Vaid
Registration number: 21BCI0014
Q2. C)

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

// CIRCULAR LINKED LIST

struct node{
    int data;
    struct node *next;
};
struct node *head = NULL;

void create(){
    do{
    int x, y;
    struct node newnode = (struct node)malloc(sizeof(struct node));
    struct node *temp;
    printf("Enter x to create a node:");
    scanf("%d", &x);
    newnode->data = x;
    newnode->next = NULL;

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

    printf("press 0 to stop.\n");
    scanf("%d", &y);

    if(y == 0){
        break;
    }

    }while(1);
}

void display(){
    struct node *temp = head;
    if(head == NULL){
        printf("CLL is empty.\n");
    }
    else{
        do{
            printf("%d-->", temp->data);
            temp = temp->next;
        }while(temp != head);
    }
    printf("Head element is: %d", temp->data);
}

void insert_at_first(){
    int x;
    struct node* newnode = (struct node*)malloc(sizeof(struct node));
    printf("Enter x to insert at first: ");
    scanf("%d", &x);
    newnode->data = x;

    if(head == NULL){
        head = newnode;
        newnode->next = head;
    }
    else{
        struct node *temp1 = head;
        while(temp1->next != head){
            temp1 = temp1->next;
        }
        newnode->next = head;
        head = newnode;
        temp1->next = head;
    }
}

void insert_at_pos(int pos){
    int x;
    struct node* newnode = (struct node*)malloc(sizeof(struct node));
    printf("Enter x to insert at position %d: ", pos);
    scanf("%d", &x);
    newnode->data = x;

    if(head == NULL){
        head = newnode;
        newnode->next = head;
    }
    else if(pos == 1){
        struct node *temp = head;
        while(temp->next != head){
            temp = temp->next;
        }
        newnode->next = head;
        head = newnode;
        temp->next = head;
    }
    else{
        struct node *temp = head;
        for(int i = 0; i< pos-2; i++){
            temp = temp->next;
        }
        if(temp->next == NULL){
            temp->next = newnode;
            newnode->next = head;
        }
        else{
            newnode->next = temp->next;
            temp->next = newnode;
        }
    }
}

void delete_first(){
    struct node *temp1 = head;
    struct node *temp2 = head;
    if(head == NULL){
        printf("CLL is empty.\n");
    }
    else{
        while(temp1->next != head){
            temp1 = temp1->next;
        }
        head = temp2->next;
        temp1->next = head;
        free(temp2);
    }
}

void delete_last(){
    struct node *temp1 = head;
    struct node *temp2;
    if(head == NULL){
        printf("CLL is empty.\n");
    }
    else{
        if(temp1->next == NULL){
            head = NULL;
            free(temp1);
        }
        while(temp1->next != head){
            temp2 = temp1;
            temp1 = temp1->next;
        }
        temp2->next = head;
        free(temp1);
    }
}

void delete_at_pos(int val){
    if(head == NULL){
        printf("CLL is empty.\n");
    }
    // finding the specific node
    struct node *temp1 = head;
    struct node *temp2;
    while(temp1->data != val){
        if(temp1->next == head){
            printf("Value is not present.\n");
            break;
        }
        temp2 = temp1;
        temp1 = temp1->next;
    }

    // check if it is only 1 node
    if (temp1->next == head)
    {
        head = NULL;
        free(temp1);
        return;
    }

    if(temp1 == head){
        temp2 = head;
        while (temp2->next != head)
```

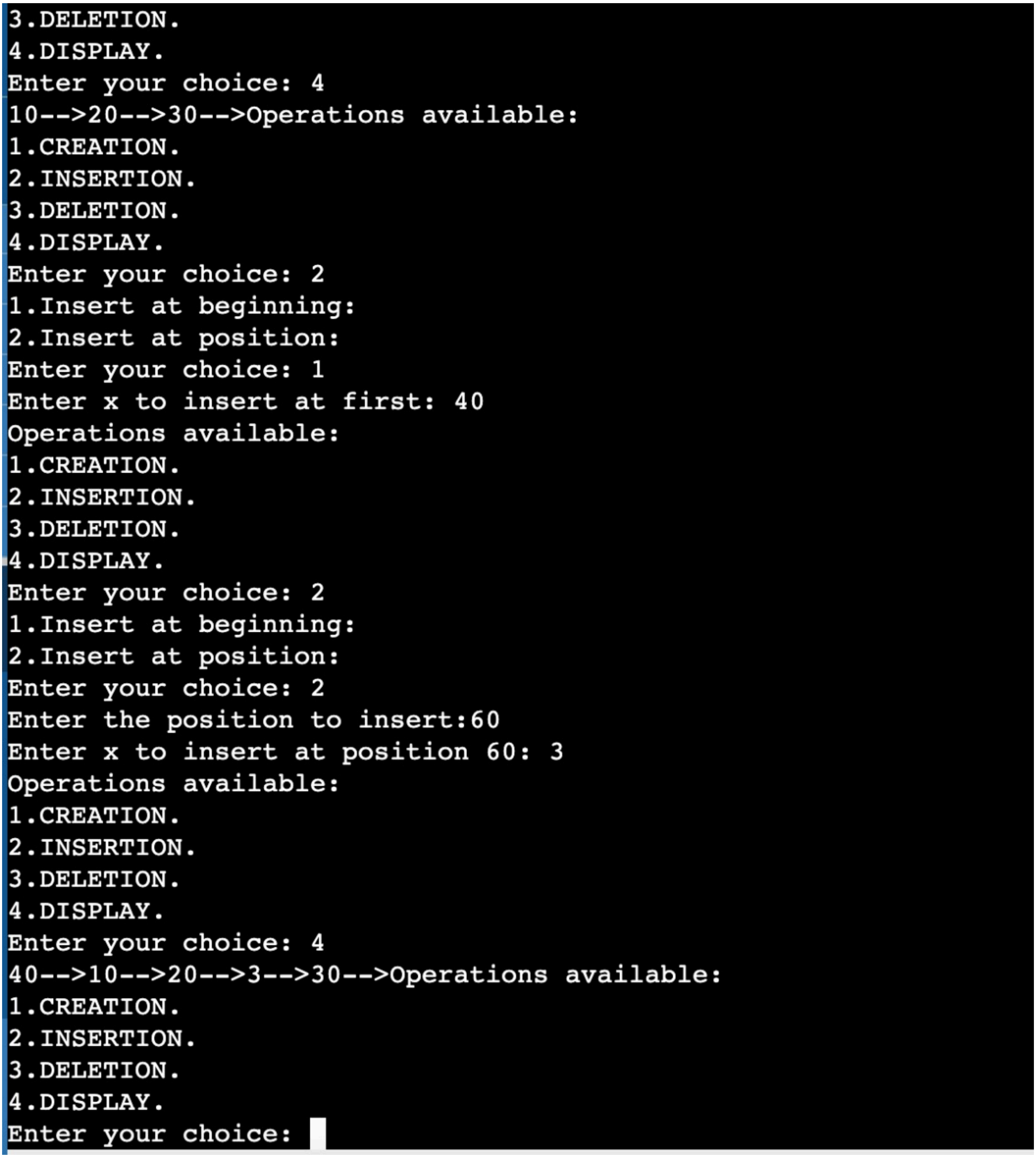
```
            temp2 = temp2->next;
        head = temp1->next;
        temp2->next = head;
        free(temp1);
    }
    else if (temp1->next == head && temp1 == head)
    {
        temp2->next = head;
        free(temp1);
    }
    else
    {
        temp2->next = temp1->next;
        free(temp1);
    }
}

int main()
{
    do{
       int ch1, ch2, pos, x, s_ele;
       printf("Operations available: \n");
       printf("1.CREATION.\n");
       printf("2.INSERTION.\n");
       printf("3.DELETION.\n");
       printf("4.DISPLAY.\n");
       printf("Enter your choice: ");
       scanf("%d", &ch1);

       switch(ch1){
           case 1:
                create();
                break;
            case 2:
                printf("1.Insert at beginning:\n");
                printf("2.Insert at position:\n");
                printf("Enter your choice: ");
                scanf("%d", &ch2);

                switch(ch2){
                    case 1:
                        insert_at_first();
                        break;
                    case 2:
                        printf("Enter the position to insert:");
                        scanf("%d", &pos);
                        insert_at_pos(pos);
                        break;
                    default:
                        exit(0);
                }
                break;
            case 3:
                 printf("1.Delete first:\n");
                printf("2.Delete last:\n");
                printf("3.Delete at position:\n");
                printf("Enter your choice: ");
                scanf("%d", &ch2);

                switch(ch2){
```

```
                case 1:
                    delete_first();
                    break;
                case 2:
                    delete_last();
                    break;
                case 3:
                    printf("Enter the element you want to delete");
                    scanf("%d", &x);
                    delete_at_pos(x);
                    break;
                default:
                    exit(0);
            }
            break;

        case 4:
            display();
            break;
        default:
            exit(0);
      }
    }while(1);

    return 0;
}
```



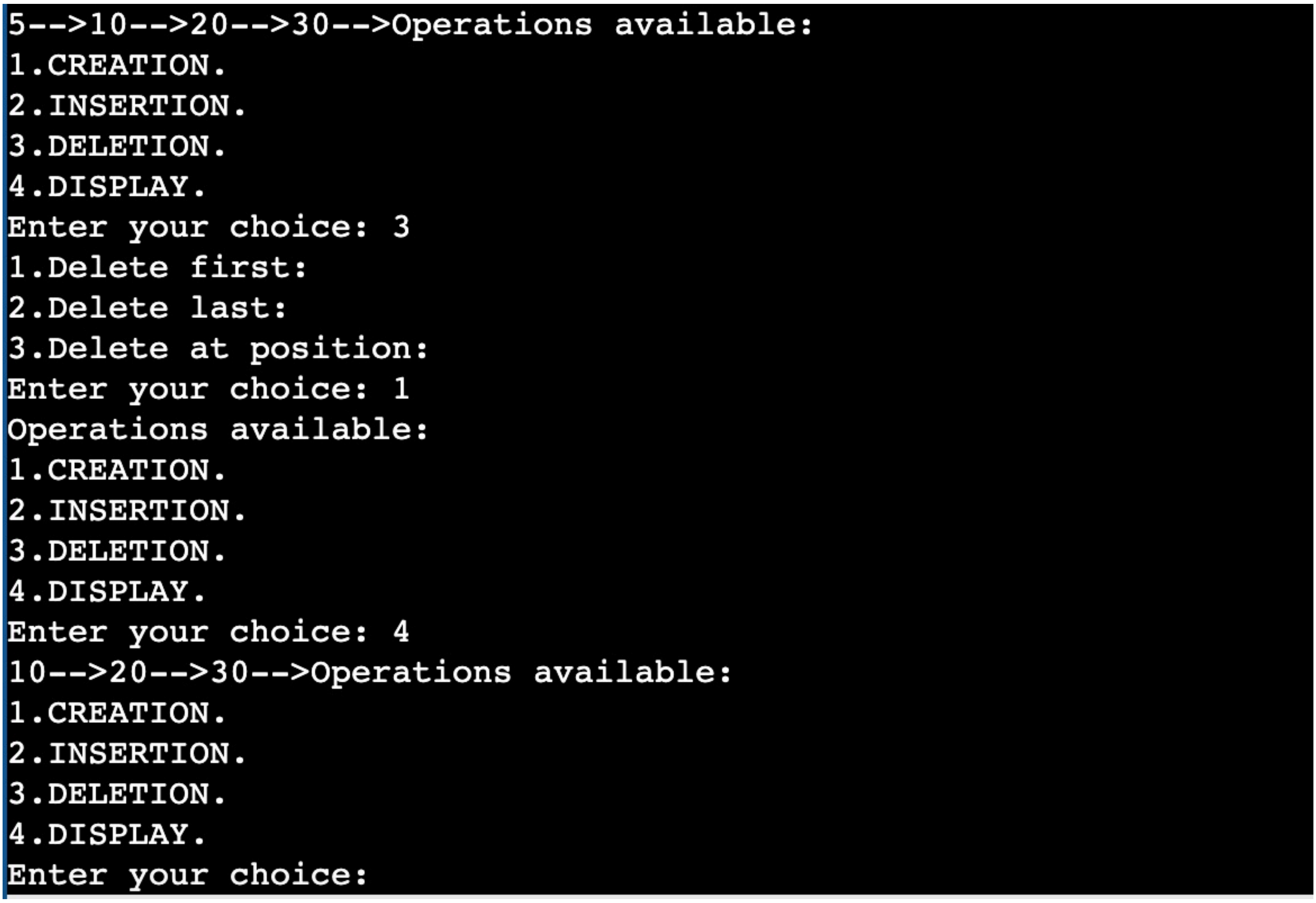
```
Operations available:
1.CREATION.
2.INSERTION.
3.DELETION.
4.DISPLAY.
Enter your choice: 1
Enter x to create a node:10
press 0 to stop.
1
Enter x to create a node:20
press 0 to stop.
1
Enter x to create a node:30
press 0 to stop.
0
Operations available:
1.CREATION.
2.INSERTION.
3.DELETION.
4.DISPLAY.
Enter your choice: 4
10-->20-->30-->Operations available:
1.CREATION.
2.INSERTION.
3.DELETION.
4.DISPLAY.
Enter your choice: 
```

Circular linked list creation



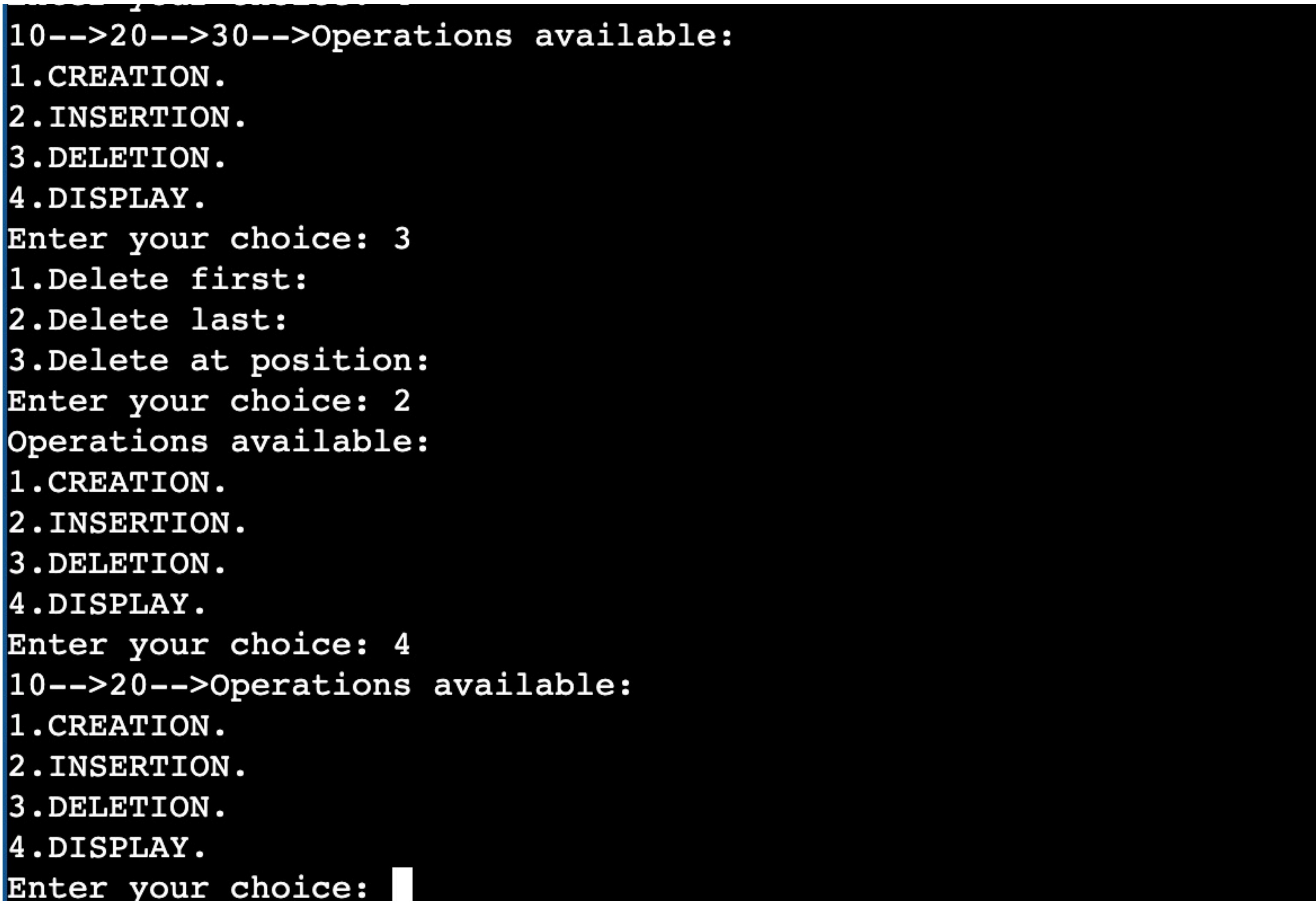
```
3.DELETION.
4.DISPLAY.
Enter your choice: 4
10-->20-->30-->Operations available:
1.CREATION.
2.INSERTION.
3.DELETION.
4.DISPLAY.
Enter your choice: 2
1.Insert at beginning:
2.Insert at position:
Enter your choice: 1
Enter x to insert at first: 40
Operations available:
1.CREATION.
2.INSERTION.
3.DELETION.
4.DISPLAY.
Enter your choice: 2
1.Insert at beginning:
2.Insert at position:
Enter your choice: 2
Enter the position to insert:60
Enter x to insert at position 60: 3
Operations available:
1.CREATION.
2.INSERTION.
3.DELETION.
4.DISPLAY.
Enter your choice: 4
40-->10-->20-->3-->30-->Operations available:
1.CREATION.
2.INSERTION.
3.DELETION.
4.DISPLAY.
Enter your choice: 
```

Circular linked list insertion



```
5-->10-->20-->30-->Operations available:
1.CREATION.
2.INSERTION.
3.DELETION.
4.DISPLAY.
Enter your choice: 3
1.Delete first:
2.Delete last:
3.Delete at position:
Enter your choice: 1
Operations available:
1.CREATION.
2.INSERTION.
3.DELETION.
4.DISPLAY.
Enter your choice: 4
10-->20-->30-->Operations available:
1.CREATION.
2.INSERTION.
3.DELETION.
4.DISPLAY.
Enter your choice:
```

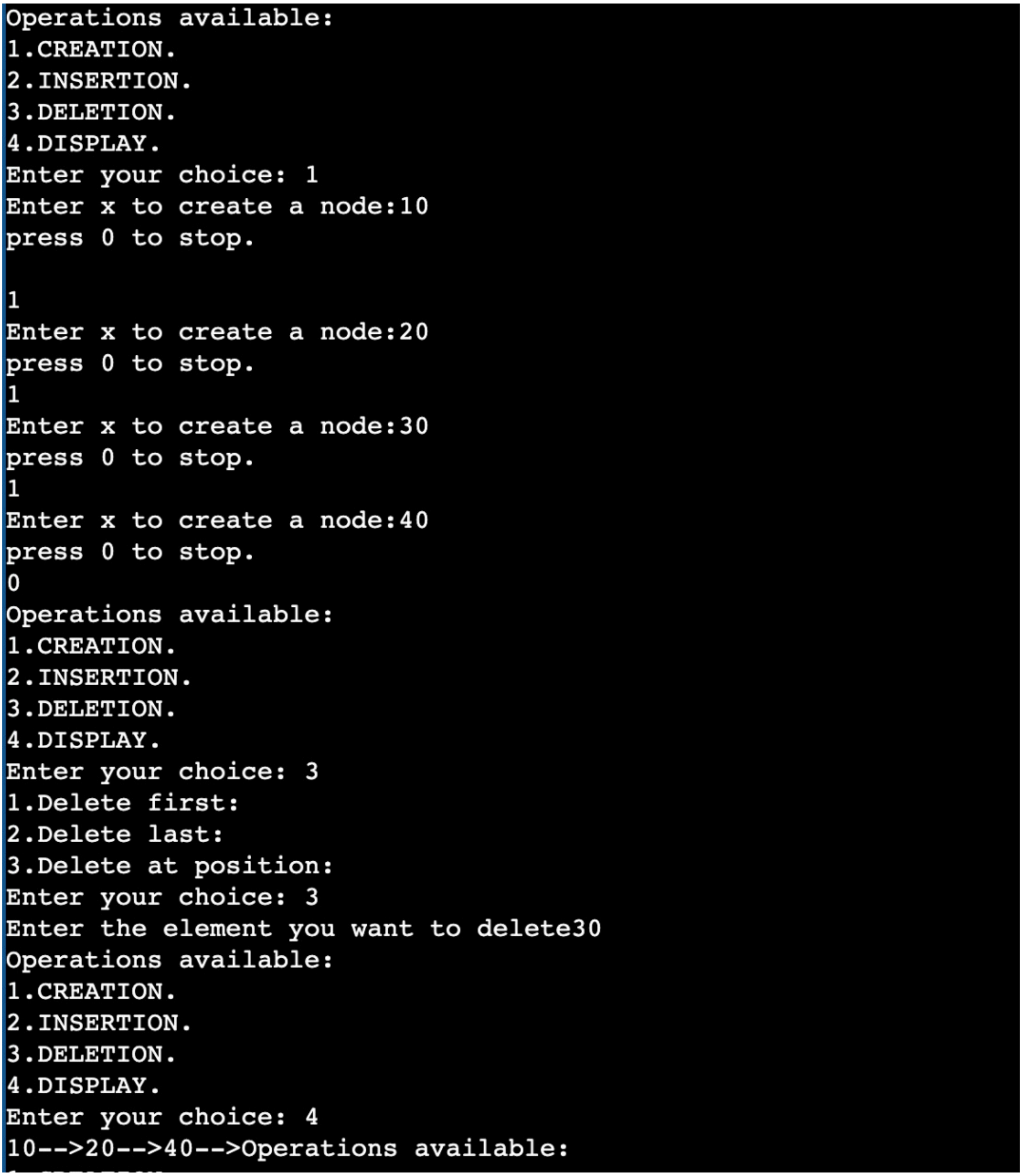
Delete first



```
Enter your choice: 4
10-->20-->30-->Operations available:
1.CREATION.
2.INSERTION.
3.DELETION.
4.DISPLAY.
Enter your choice: 3
1.Delete first:
2.Delete last:
3.Delete at position:
Enter your choice: 2
Operations available:
1.CREATION.
2.INSERTION.
3.DELETION.
4.DISPLAY.
Enter your choice: 4
10-->20-->Operations available:
1.CREATION.
2.INSERTION.
3.DELETION.
4.DISPLAY.
Enter your choice: 
```

Delete last

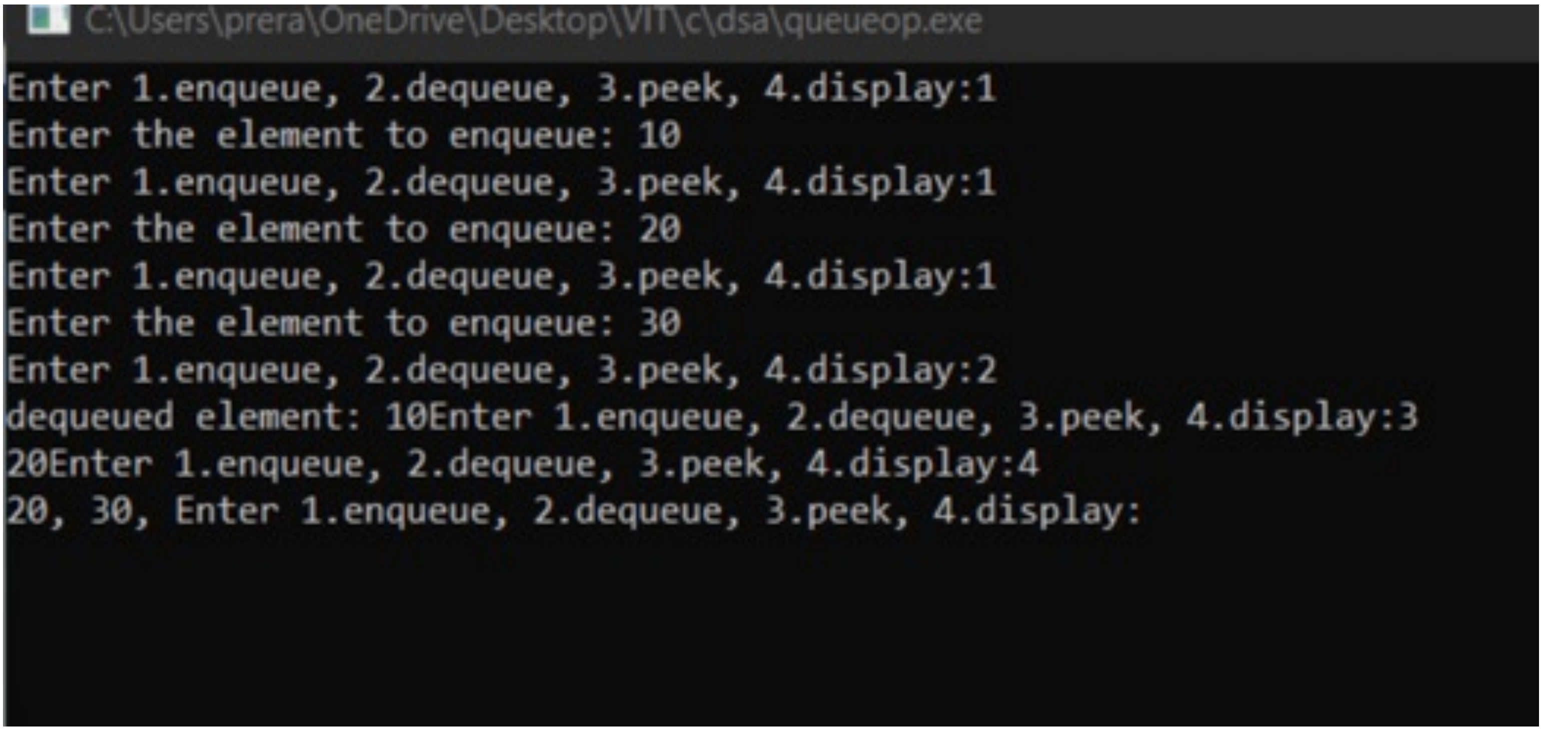
Delete at position



```
Operations available:
1.CREATION.
2.INSERTION.
3.DELETION.
4.DISPLAY.
Enter your choice: 1
Enter x to create a node:10
press 0 to stop.

1
Enter x to create a node:20
press 0 to stop.
1
Enter x to create a node:30
press 0 to stop.
1
Enter x to create a node:40
press 0 to stop.
0
Operations available:
1.CREATION.
2.INSERTION.
3.DELETION.
4.DISPLAY.
Enter your choice: 3
1.Delete first:
2.Delete last:
3.Delete at position:
Enter your choice: 3
Enter the element you want to delete30
Operations available:
1.CREATION.
2.INSERTION.
3.DELETION.
4.DISPLAY.
Enter your choice: 4
10-->20-->40-->Operations available:
```

Q1.
Queue implementation

```
#include<stdio.h>
#include <stdlib.h>
int queue[10];
int front=-1;
int rear=-1;
void enqueue(int x){
if(rear==9){
printf("overflow");
}
else if(front==-1&&rear==-1){
front=rear=0;
queue[rear]=x;
}
else{
rear++;
queue[rear]=x;
}
}
void dequeue(){
if(front==-1&&rear==-1){
printf("underflow");
}
else if(front==rear){
front=rear=-1;
}
else{
printf("dequeued element: %d",queue[front]);
front++;
}
}
void display(){
if(front==-1&&rear==-1){
printf("empty");
}
else{
int i;
for(i=front;i<rear+1;i++){
   printf("%d, ",queue[i]);
}
}
}
void peek(){
if(front==-1&&rear==-1){
printf("empty");
}
else{
printf("%d",queue[front]);
}
}

int main()
{
do{int n, ch;
printf("Enter 1.enqueue, 2.dequeue, 3.peek, 4.display:");
scanf("%d", &ch);
switch(ch){
case 1:
printf("Enter the element to enqueue: ");
scanf("%d", &n);
enqueue(n);
break;
case 2:
dequeue();
break;
case 3:
peek();
break;
case 4:
display();
break;
default:
exit(0);
}
}while(1);
return 0;
}
```



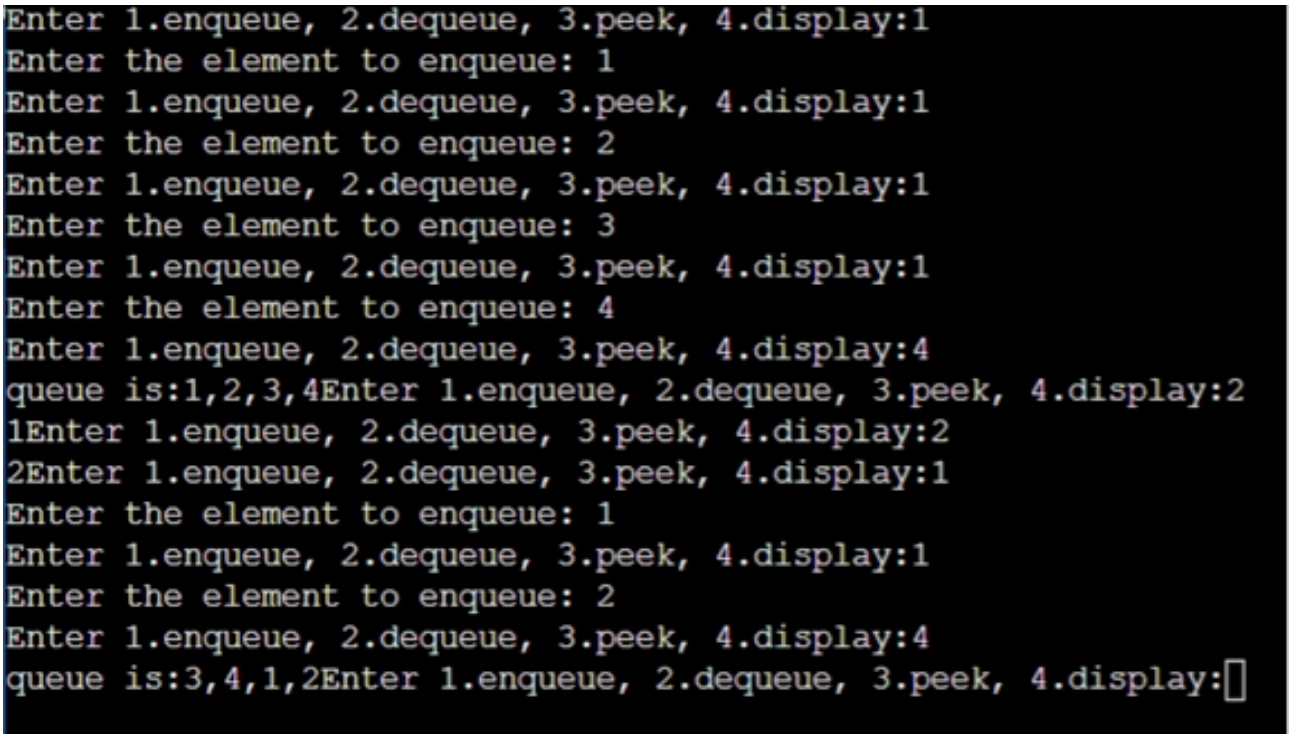
C:\Users\prera\OneDrive\Desktop\VIT\c\dsa\queueop.exe

```
Enter 1.enqueue, 2.dequeue, 3.peek, 4.display:1
Enter the element to enqueue: 10
Enter 1.enqueue, 2.dequeue, 3.peek, 4.display:1
Enter the element to enqueue: 20
Enter 1.enqueue, 2.dequeue, 3.peek, 4.display:1
Enter the element to enqueue: 30
Enter 1.enqueue, 2.dequeue, 3.peek, 4.display:2
dequeued element: 10Enter 1.enqueue, 2.dequeue, 3.peek, 4.display:3
20Enter 1.enqueue, 2.dequeue, 3.peek, 4.display:4
20, 30, Enter 1.enqueue, 2.dequeue, 3.peek, 4.display:
```

Circular Queue Implementation

```
#include<stdio.h>
#include<stdlib.h>
int queue[5];
int front=-1;
int rear=-1;
void enqueue(int x){
if(front==-1&&rear==-1){
front=rear=0;
queue[rear]=x;
}
else if((rear+1)%4==front){
printf("queue full");
}
else{
rear=(rear+1)%5;
queue[rear]=x;
}
}
void dequeue(){
if(front==-1&&rear==-1){
printf("empty queue");
}
else if(front==rear){
front=rear=-1;
}
else{
printf("%d",queue[front]);
front=(front+1)%5;
}
}
void display(){
int i=front;
if(front==-1&&rear==-1){
printf("empty queue");
}

else{
printf("queue is:");
while(i!=rear){
printf("%d,",queue[i]);
i=(i+1)%5;
}
printf("%d",queue[rear]);
}
}
void peek(){
if(front==-1&&rear==-1){
printf("queue is empty\n");
}
else
{
printf("peek: %d",queue[front])

}
}
int main()
{
do{
```

```
int n, ch;
printf("Enter 1.enqueue, 2.dequeue, 3.peek, 4.display:");
scanf("%d", &ch);
switch(ch){
case 1:
printf("Enter the element to enqueue: ");
scanf("%d", &n);
enqueue(n);
break;
case 2:
dequeue();
break;
case 3:
peek();
break;

case 4:
display();
break;

default:
exit(0);
}
}while(1);
return 0;
```



```
Enter 1.enqueue, 2.dequeue, 3.peek, 4.display:1
Enter the element to enqueue: 1
Enter 1.enqueue, 2.dequeue, 3.peek, 4.display:1
Enter the element to enqueue: 2
Enter 1.enqueue, 2.dequeue, 3.peek, 4.display:1
Enter the element to enqueue: 3
Enter 1.enqueue, 2.dequeue, 3.peek, 4.display:1
Enter the element to enqueue: 4
Enter 1.enqueue, 2.dequeue, 3.peek, 4.display:4
queue is:1,2,3,4Enter 1.enqueue, 2.dequeue, 3.peek, 4.display:2
1Enter 1.enqueue, 2.dequeue, 3.peek, 4.display:2
2Enter 1.enqueue, 2.dequeue, 3.peek, 4.display:1
Enter the element to enqueue: 1
Enter 1.enqueue, 2.dequeue, 3.peek, 4.display:1
Enter the element to enqueue: 2
Enter 1.enqueue, 2.dequeue, 3.peek, 4.display:4
queue is:3,4,1,2Enter 1.enqueue, 2.dequeue, 3.peek, 4.display:
```

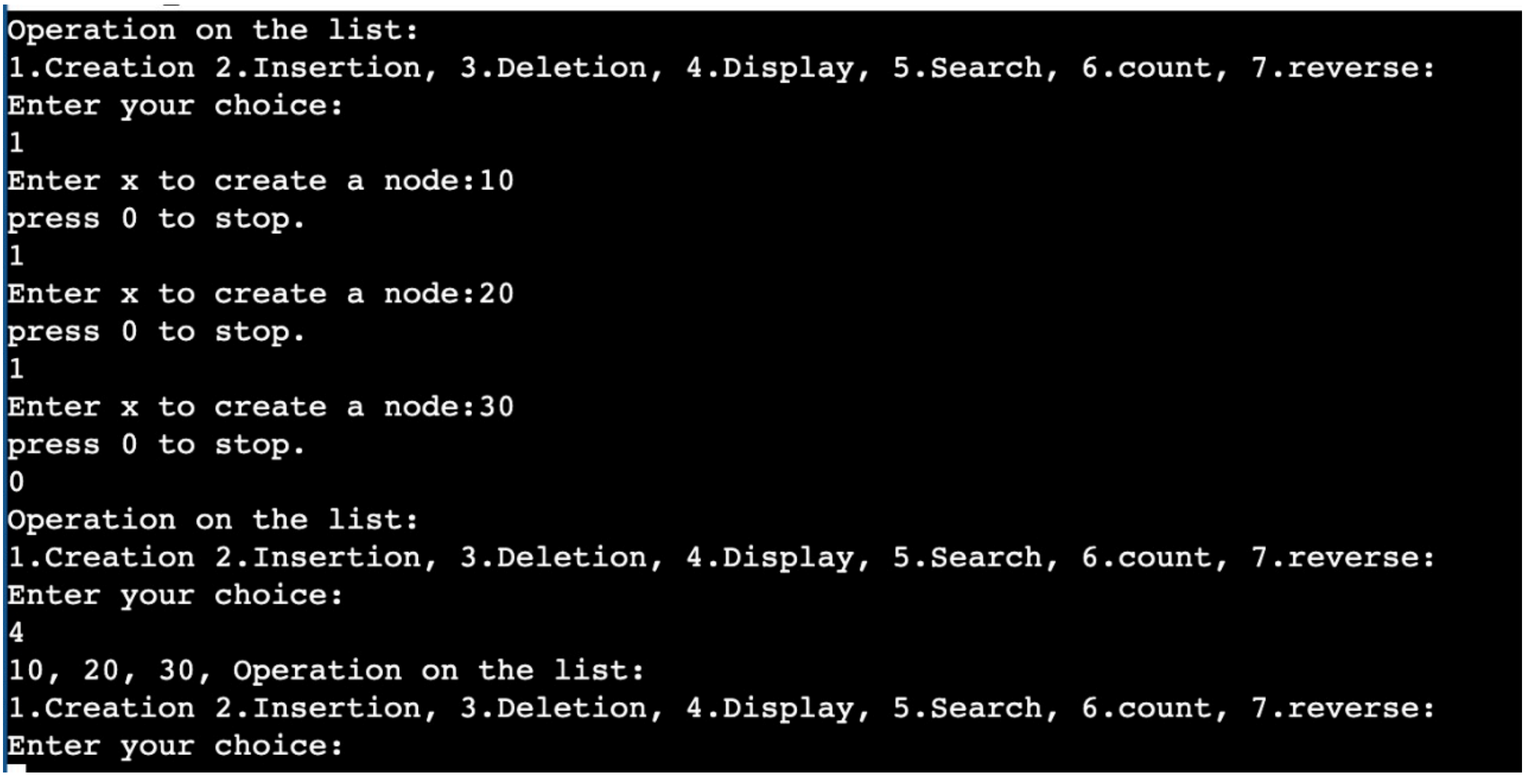
```
}
```

IMPLEMENTATION OF LINKED LISTS

```
#include <stdio.h>
#include <stdlib.h>
void display();

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

struct node *head = NULL;

void create(){
    do{
    int x, y;
    struct node newnode = (struct node)malloc(sizeof(struct node));
    struct node *temp;
    printf("Enter x to create a node:");
    scanf("%d", &x);
    newnode->data = x;
    newnode->next = NULL;

    if(head == NULL){
        head = temp = newnode;
    }
    else{
        temp->next = newnode;
        temp = newnode;
    }
    printf("press 0 to stop.\n");
    scanf("%d", &y);

    if(y == 0){
        break;
    }

    }while(1);
}

void count(){
    int count = 0;
    if(head == NULL){
        printf("Linked list is empty.\n");
    }
    else{
        struct node *temp = head;
        while(temp != NULL){
            count++;
            temp = temp->next;
        }
    }
    printf("There are %d elements in the linked list.\n", count);
}

void reverse(){
    struct node *temp1 = NULL;
    struct node *temp2 = NULL;
    struct node *current = head;
    while(current != NULL) {
        temp1 = current->next;
        current->next = temp2;
        temp2 = current;
        current = temp1;
    }
    head = temp2;
}

void insert_first(){
    struct node newnode = (struct node *)malloc(sizeof(struct node));
    int x;
    printf("Enter int to insert first linked list: ");
    scanf("%d", &x);

    if(head == NULL){
        printf("The linked list is empty.\n");
        newnode->data = x;
        newnode->next = NULL;
        head = newnode;
    }
    else{
        newnode->data = x;
        newnode->next = head;
        head = newnode;
    }
}

void insert_last(){
    struct node newnode = (struct node *)malloc(sizeof(struct node));

    int x;
    printf("Enter int to insert last in linked list: ");
    scanf("%d", &x);
    newnode->data = x;

    if(head == NULL){
        printf("LL is empty.\n");
        newnode->next = NULL;
        head = newnode;
    }
    else{
        struct node *p = head;
        while(p->next != NULL){
            p = p->next;
        }
        newnode->next = NULL;
        p->next = newnode;
    }
}

void insert_pos(int pos){
    struct node newnode = (struct node)malloc(sizeof(struct node*));
    struct node *temp = head;
    int x;
    printf("Enter x:");
    scanf("%d", &x);

    if(pos == 1){ // inserting element as 1st element
        newnode->data = x;
        newnode->next = NULL;
        head = newnode;
    }
    else{
        for(int i = 0; i<pos-2;i++){
            temp = temp->next;
        }
        if(temp->next == NULL){
            // insert_last();
            newnode->next = NULL;
            temp->next = newnode;
        }
        else{
            newnode->data = x;
            newnode->next = temp->next;
            temp->next = newnode;
        }

    }
}

void display(){
    struct node *temp = head;
    if(head == NULL){
        printf("List is empty\n");
    }
    else{
        while(temp != NULL){
        printf("%d, ", temp->data);
        temp = temp->next;
        }
    }

}

void delete_first(){
    if(head == NULL){
        printf("Linked list is empty.\n");
    }
    else{
        struct node *temp = head;
        head = temp->next;
        free(temp);
    }
}

void delete_last(){
    struct node *temp1 = head;
    struct node *temp2;
    if(head == NULL){
        printf("List is empty.\n");
    }
    else if(temp1->next == NULL){
        head = temp1->next;
        free(temp1);
    }
    else{
        while(temp1->next != NULL){
            temp2 = temp1;
            temp1 = temp1->next;
        }
        temp2->next = NULL;
        free(temp1);
    }
}

void delete_position(int pos){
    struct node *temp1 = head;
    struct node *temp2;

    if(head == NULL){
        printf("List is empty\n");
    }
    else{
        if(pos == 1){
            head = temp1->next;
            free(temp1);
        }
        else{
            for(int i = 0; i<pos-2; i++){
                temp1 = temp1->next;
            }
            temp2 = temp1->next;
            temp1->next = temp2->next;
            free(temp2);
        }
    }
}

void search(){
    int x;
    printf("Enter the element to search:");
    scanf("%d", &x);
    struct node *temp = head;
    int pos = 1;
    if(head == NULL){
        printf("List is empty.\n");
    }
    else{
        while(temp != NULL){
            if(temp->data == x){
                printf("The position of the element is: %d", pos);
                break;
            }
            // else{
            //     printf("ELEMENT NOT FOUND");
            // }
        temp = temp->next;
        pos++;
        }
        if(temp == NULL){
            printf("ELEMENT NOT FOUND.\n");
        }
    }
}

int main()
{
    int ch;
    do{
```

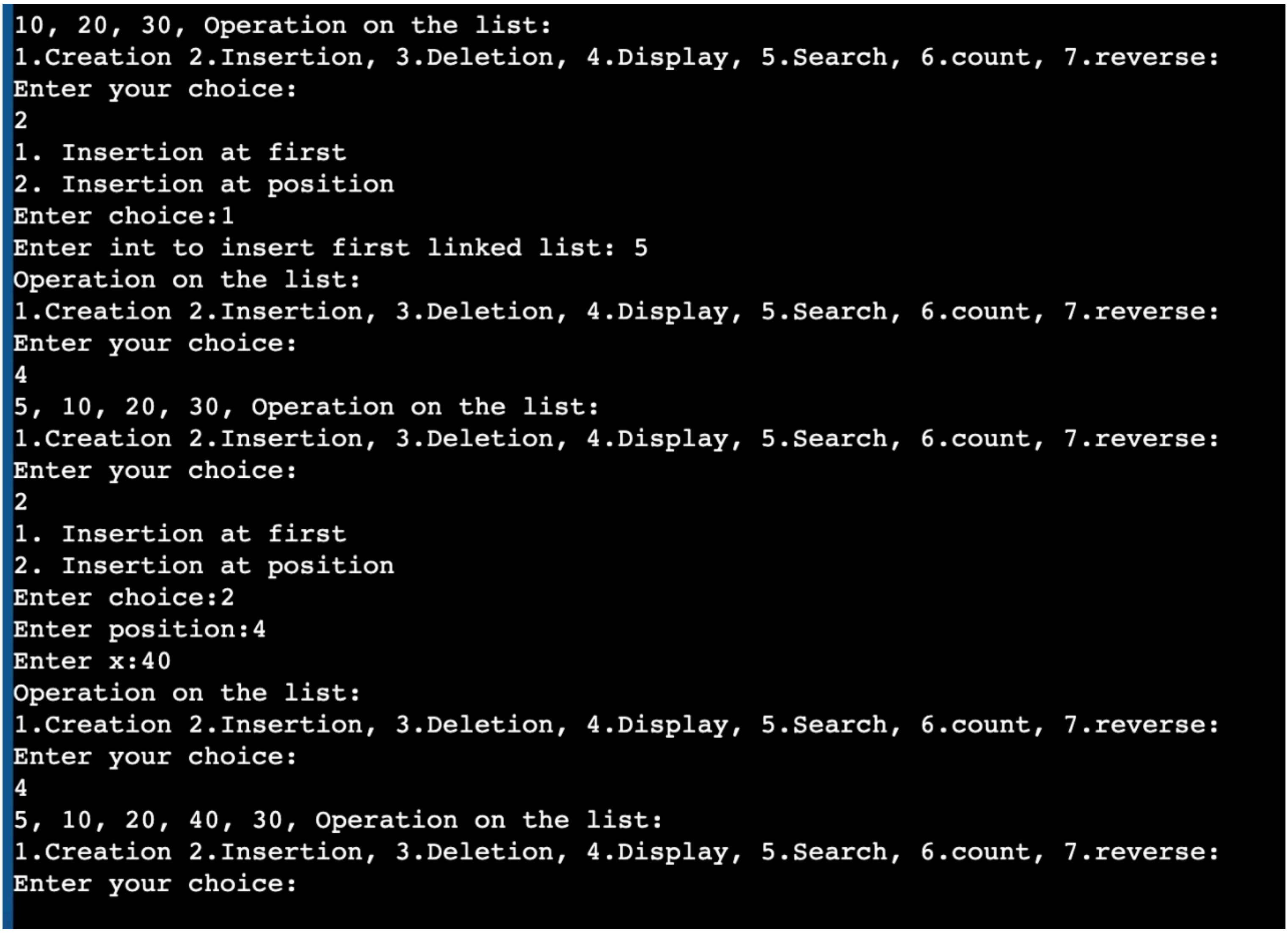
```
printf("Operation on the list:\n");
printf("1.Creation 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.count, 7.reverse:\n");
printf("Enter your choice:\n");
scanf("%d", &ch);
int pls, pos;
switch(ch){
    case 1:
        create();
        break;
    case 2:
        printf("1. Insertion at first\n");
        printf("2. Insertion at position\n");
        printf("Enter choice:");
        scanf("%d", &pls);

        switch(pls){
            case 1:
                insert_first();
                break;
            case 2:
                printf("Enter position:");
                scanf("%d", &pos);
                insert_pos(pos);
                break;
            default:
                exit(0);
        }
        break;
    case 3:
        printf("1. Deletion at first\n");
        printf("2. Deletion at last\n");
        printf("3. Deletion at position\n");
        printf("Enter choice:");
        scanf("%d", &pls);
        switch(pls){
            case 1:
                delete_first();
                printf("After deletion:\n");
                display();
                break;
            case 2:
                delete_last();
                printf("After deletion:\n");
                display();
                break;
            case 3:
                printf("Enter position:");
                scanf("%d", &pos);
                delete_position(pos);
                printf("After deletion:\n");
                display();
                break;
            default:
                exit(0);
        }
        break;
    case 4:
        display();
        break;
    case 5:
```

```
                    search();
                    break;
                case 6:
                    count();
                    break;
                case 7:
                    reverse();
                    break;
                default:
                    exit(0);
            }
        }while(1);
        return 0;
    }
```



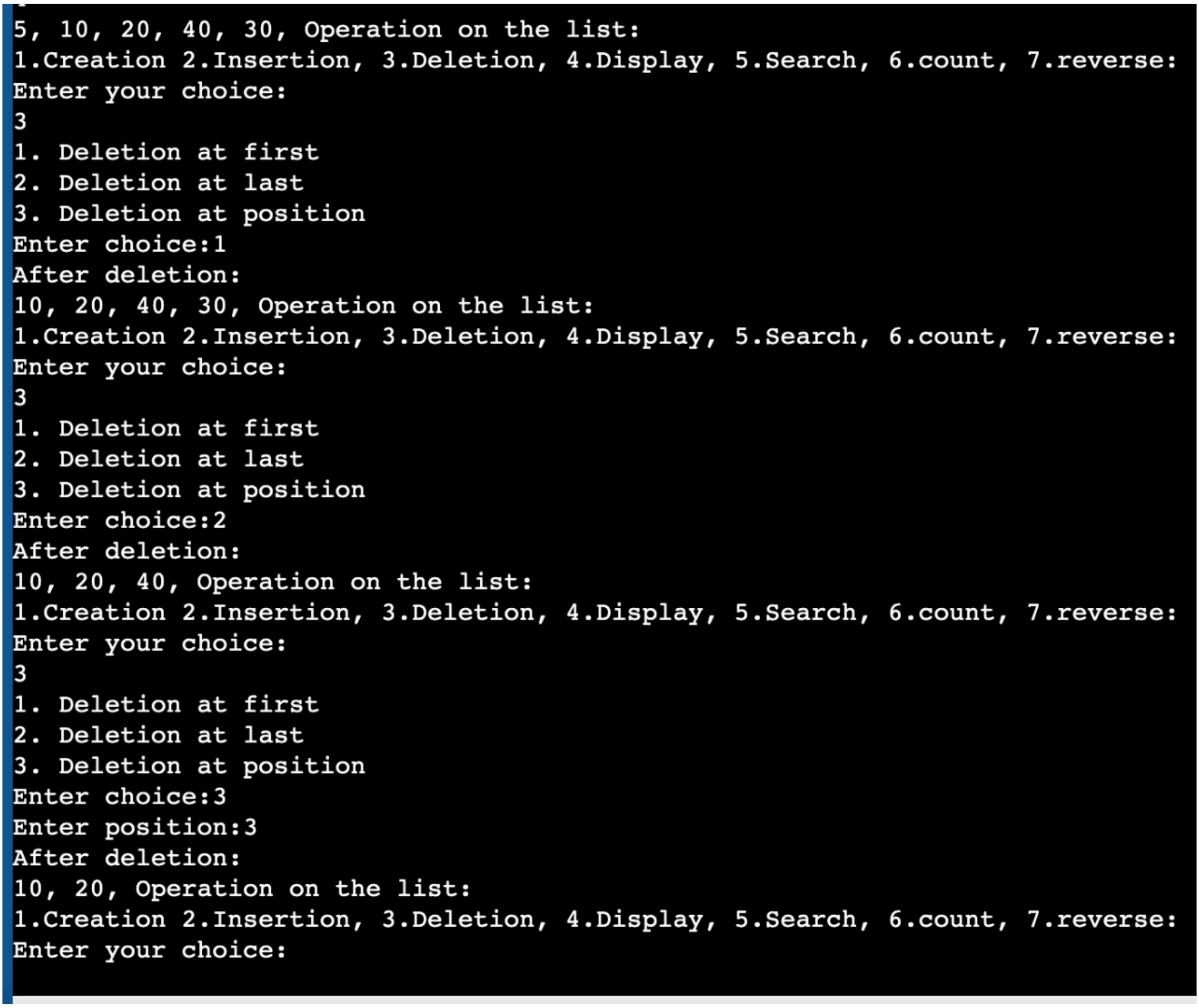
```
Operation on the list:
1.Creation 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.count, 7.reverse:
Enter your choice:
1
Enter x to create a node:10
press 0 to stop.
1
Enter x to create a node:20
press 0 to stop.
1
Enter x to create a node:30
press 0 to stop.
0
Operation on the list:
1.Creation 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.count, 7.reverse:
Enter your choice:
4
10, 20, 30, Operation on the list:
1.Creation 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.count, 7.reverse:
Enter your choice:
```

Linked List Creation



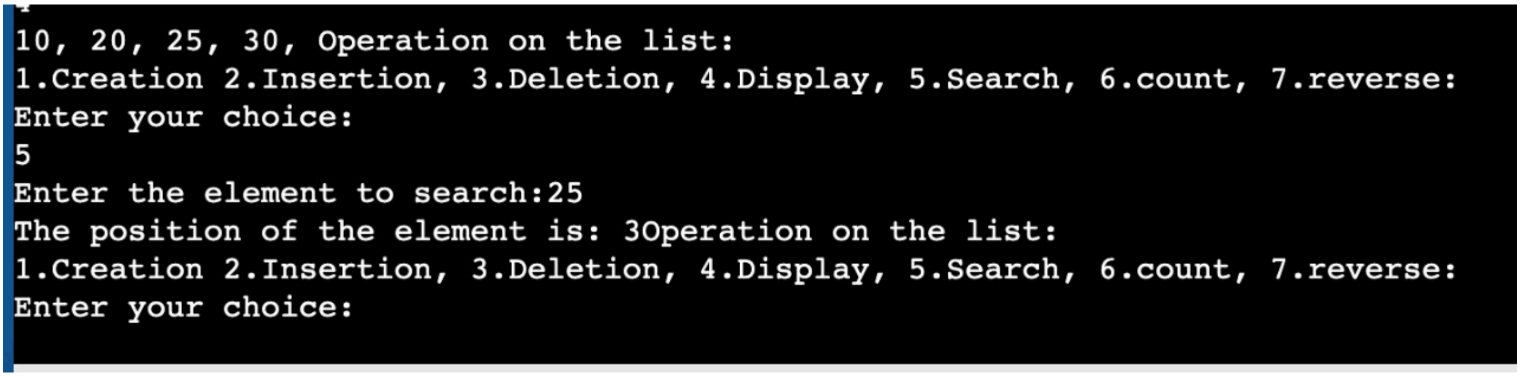
```
10, 20, 30, Operation on the list:
1.Creation 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.count, 7.reverse:
Enter your choice:
2
1. Insertion at first
2. Insertion at position
Enter choice:1
Enter int to insert first linked list: 5
Operation on the list:
1.Creation 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.count, 7.reverse:
Enter your choice:
4
5, 10, 20, 30, Operation on the list:
1.Creation 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.count, 7.reverse:
Enter your choice:
2
1. Insertion at first
2. Insertion at position
Enter choice:2
Enter position:4
Enter x:40
Operation on the list:
1.Creation 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.count, 7.reverse:
Enter your choice:
4
5, 10, 20, 40, 30, Operation on the list:
1.Creation 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.count, 7.reverse:
Enter your choice:
```

Linked list insertion



```
5, 10, 20, 40, 30, Operation on the list:
1.Creation 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.count, 7.reverse:
Enter your choice:
3
1. Deletion at first
2. Deletion at last
3. Deletion at position
Enter choice:1
After deletion:
10, 20, 40, 30, Operation on the list:
1.Creation 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.count, 7.reverse:
Enter your choice:
3
1. Deletion at first
2. Deletion at last
3. Deletion at position
Enter choice:2
After deletion:
10, 20, 40, Operation on the list:
1.Creation 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.count, 7.reverse:
Enter your choice:
3
1. Deletion at first
2. Deletion at last
3. Deletion at position
Enter choice:3
Enter position:3
After deletion:
10, 20, Operation on the list:
1.Creation 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.count, 7.reverse:
Enter your choice:
```

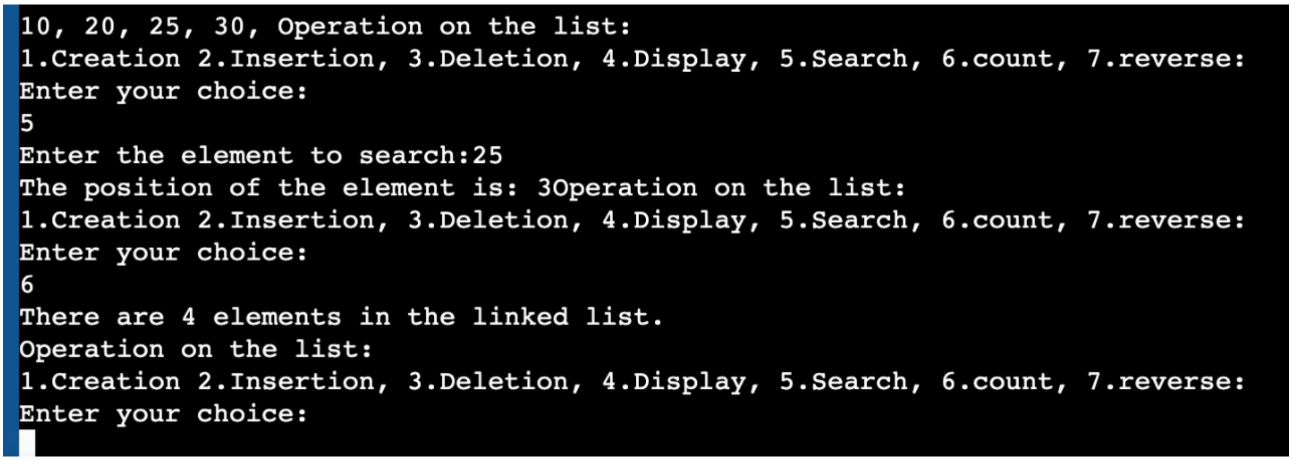
Linked list deletion



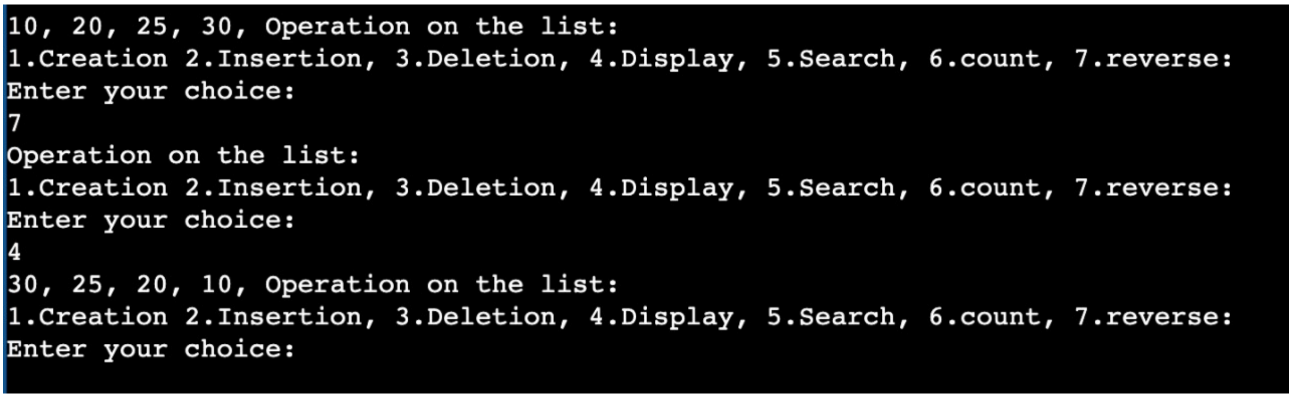
```
10, 20, 25, 30, Operation on the list:
1.Creation 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.count, 7.reverse:
Enter your choice:
5
Enter the element to search:25
The position of the element is: 3Operation on the list:
1.Creation 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.count, 7.reverse:
Enter your choice:
```

Linked list search

linked list count



```
10, 20, 25, 30, Operation on the list:
1.Creation 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.count, 7.reverse:
Enter your choice:
5
Enter the element to search:25
The position of the element is: 3Operation on the list:
1.Creation 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.count, 7.reverse:
Enter your choice:
6
There are 4 elements in the linked list.
Operation on the list:
1.Creation 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.count, 7.reverse:
Enter your choice:
```



```
10, 20, 25, 30, Operation on the list:
1.Creation 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.count, 7.reverse:
Enter your choice:
7
Operation on the list:
1.Creation 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.count, 7.reverse:
Enter your choice:
4
30, 25, 20, 10, Operation on the list:
1.Creation 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.count, 7.reverse:
Enter your choice:
```

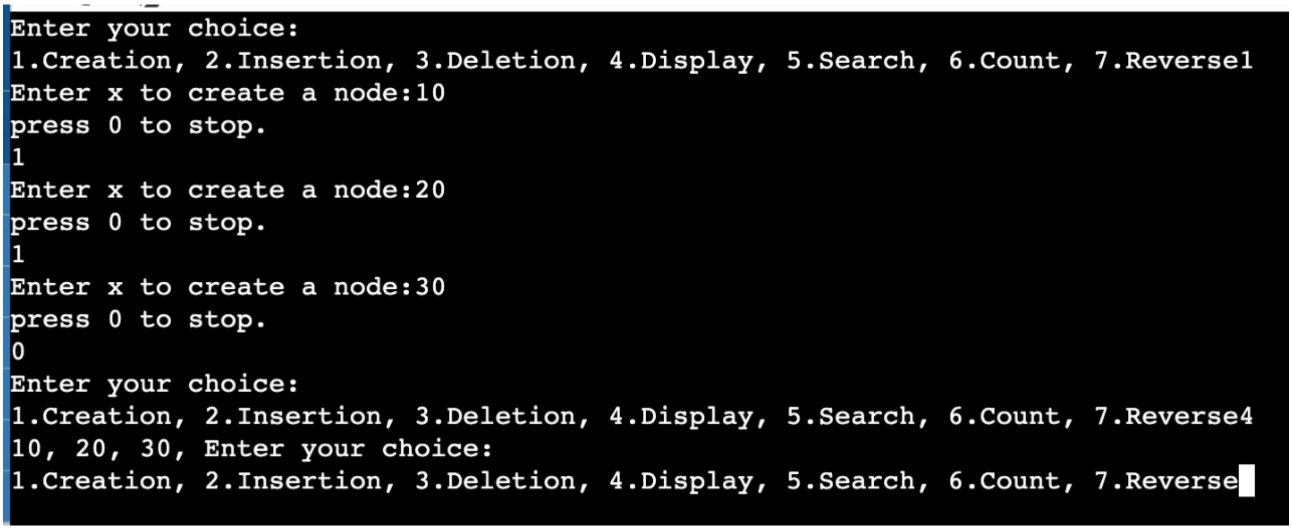
Linked list reverse

Q2. B double linked list

```
#include <stdio.h>
#include <stdlib.h>
//DOUBLY LINKED LIST
struct node{
    int data;
    struct node *prev;
    struct node *next;
};

struct node* head = NULL;

void creation(){
    do{
        int x, y;
        struct node newnode = (struct node)malloc(sizeof(struct node));
        struct node *temp;
        printf("Enter x to create a node:");
        scanf("%d", &x);
        newnode->data = x;
        newnode->next = NULL;
        newnode->prev = NULL;

        if(head == NULL){
            head = temp = newnode;
        }
        else{
            newnode->next = NULL;
            temp->next = newnode;
            newnode->prev = temp;
            temp = newnode;
        }
        printf("press 0 to stop.\n");
        scanf("%d", &y);

        if(y == 0){
            break;
        }
    }while(1);
}

void count(){
    struct node *temp = head;
    int count = 0;
    if(head == NULL){
        printf("DLL is empty.\n");
    }
    else{
        while(temp != NULL){
            count++;
            temp = temp->next;
        }
    }
    printf("The number of element in the DLL is: %d", count);
}

void reverse(){
    struct node* temp1 = NULL;
    struct node* current = head;
    while (current != NULL) {
        temp1 = current->prev;
        current->prev = current->next;
        current->next = temp1;
        current = current->prev;
    }

    if (temp1 != NULL){
        head = temp1->prev;
    }
}

void insert_first(){
    struct node* newnode = (struct node*)malloc(sizeof(struct node));
    int x;
    printf("Enter the element to insert at first:");
    scanf("%d", &x);

    if(head == NULL){
        newnode->data = x;
        newnode->next = NULL;
        newnode->prev = NULL;
        head = newnode;
    }
    else{
        struct node* temp = head;
        newnode->data = x;
        newnode->next = temp;
        newnode->prev = NULL;
        temp->prev = newnode;
        head = newnode;
    }
}

void insert_at_pos(int pos){
    struct node newnode = (struct node)malloc(sizeof(struct node));
    struct node* temp = head;
    int x;
    printf("Enter the element to insert at position:");
    scanf("%d", &x);

    newnode->data = x;

    if(head == NULL){
        newnode->next = NULL;
        newnode->prev = NULL;
        head = newnode;
    }
    else if(pos == 1){
        newnode->next = temp;
        newnode->prev = NULL;
        temp->prev = newnode;
        head = newnode;
    }
    else{
        for(int i = 0; i<pos-2; i++){
            temp = temp->next;
        }
        if(temp->next == NULL){
            // insert_last();
            newnode->next = NULL;
            newnode->prev = temp;
            temp->next = newnode;
        }
        else{
            newnode->next = temp->next;
            temp->next->prev = newnode;
            temp->next = newnode;
            newnode->prev = temp;
        }
    }
}

void delete_first(){
    struct node *temp = head;
    if(head == NULL){
        printf("Doubly Linked List is empty.\n");
    }
    else if(head->next == NULL){
        head = NULL;
        free(temp);
    }
    else{
        head = temp->next;
        temp->next->prev = NULL; // head->prev = NULL
        free(temp);
    }
}

void delete_last(){
    struct node* temp1 = head;
    struct node* temp2;
    if(head == NULL){
        printf("DLL is empty.\n");
    }
    else if(head->next == NULL){
        head = NULL;
        free(temp1);
    }
    else{
        while(temp1->next != NULL){
            temp2 = temp1;
            temp1 = temp1->next;
        }
        // temp2 = temp1->prev;
        temp2->next = NULL;
        free(temp1);
    }
}

void delete_at_pos(int pos){
    struct node *temp1 = head;
    struct node *temp2;

    if(head == NULL){
        printf("DLL is empty.\n");
    }
    else{
        if(pos == 1){
            head = temp1->next;
            head->prev = NULL;
            free(temp1);
        }
        else{
            for(int i = 0; i<pos-2; i++){
                temp1 = temp1->next;
            }
            temp2 = temp1->next;
            if(temp2->next == NULL){
                delete_last();
                // temp1->next = NULL;
                // free(temp2);
            }
            temp1->next = temp2->next;
            temp2->next->prev = temp1;
            free(temp2);
        }
    }
}

void display(){
    if(head == NULL){
        printf("DLL is empty.\n");
    }
    else{
        struct node *temp = head;
        while(temp!=NULL){
            printf("%d, ", temp->data);
            temp = temp->next;
        }
    }
}

void search(int s_ele){
    struct node *temp = head;

    while(temp!=NULL){
        if(temp->data == s_ele){
            break;
        }
        temp = temp->next;
    }

    if(temp != NULL){
        printf("Elemet is present.\n");
    }
    else{
        printf("Not present.\n");
    }
}

int main()
{
    do{
        int ch1, ch2, s_ele, pos;
        printf("Enter your choice:\n");
        printf("1.Creation, 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.Count, 7.Reverse");
        scanf("%d", &ch1);

        switch(ch1){
```

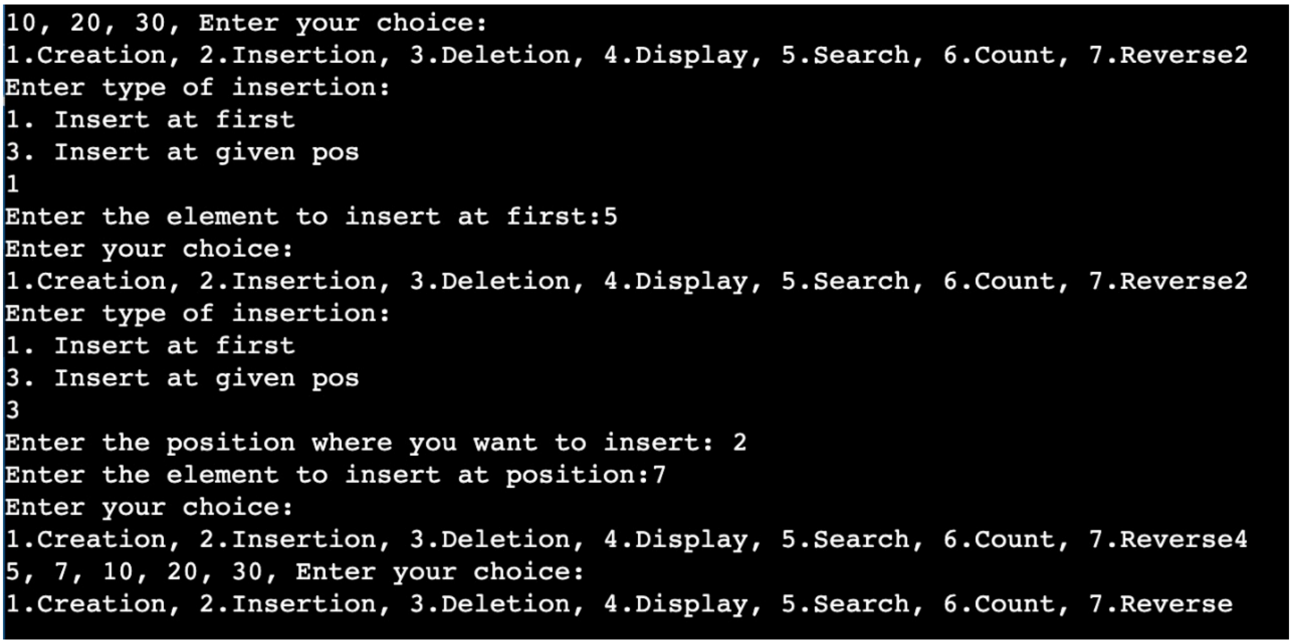
```
case 1:
   creation();
   break;
case 2:
   printf("Enter type of insertion:\n");
   printf("1. Insert at first\n");
   // printf("2. Insert at last\n");
   printf("3. Insert at given pos\n");
   scanf("%d", &ch2);

   switch(ch2){
      case 1:
         insert_first();
         break;
      case 3:
         printf("Enter the position where you want to insert: ");
         scanf("%d", &pos);
         insert_at_pos(pos);
         break;
      default:
         exit(0);
   }
   break;

case 3:
   printf("Enter type of deletion:\n");
   printf("1. Delete first\n");
   printf("2. Delete last\n");
   printf("3. Delete at given pos\n");
   scanf("%d", &ch2);

   switch(ch2){
      case 1:
         delete_first();
         break;
      case 2:
         delete_last();
         break;
      case 3:
         printf("Enter the position of the node you want ot delete: \n");
         scanf("%d", &pos);
         delete_at_pos(pos);
         break;
      default:
         exit(0);
   }
   break;

case 4:
   display();
   break;

case 5:
   printf("Enter element you want to search: ");
   scanf("%d", &s_ele);
   search(s_ele);
   break;

case 6:
   count();
```

```
                break;

            case 7:
                reverse();
                break;

            default:
                exit(0);
        }
    }while(1);

    return 0;
}
```



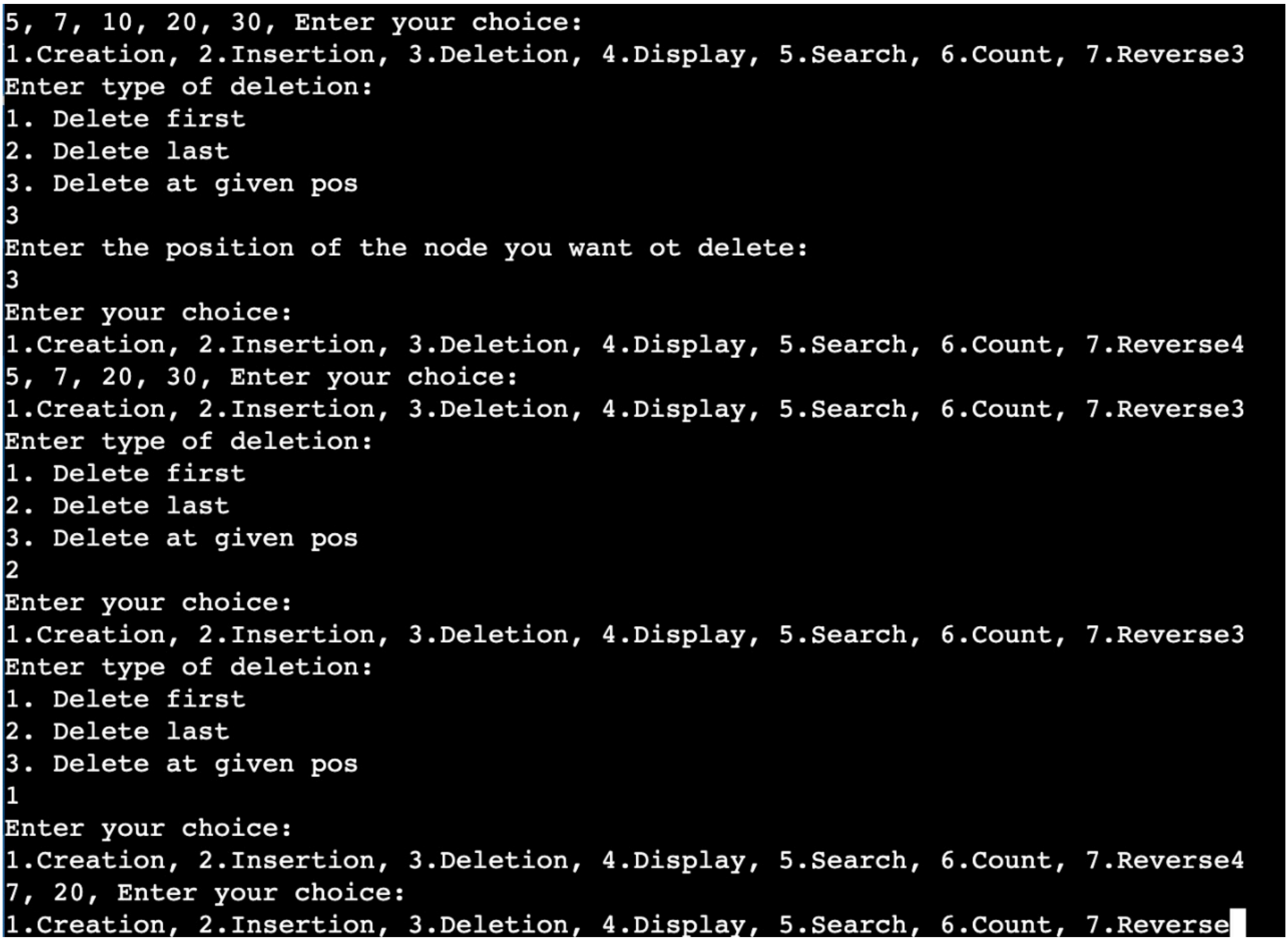
```
Enter your choice:
1.Creation, 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.Count, 7.Reverse1
Enter x to create a node:10
press 0 to stop.
1
Enter x to create a node:20
press 0 to stop.
1
Enter x to create a node:30
press 0 to stop.
0
Enter your choice:
1.Creation, 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.Count, 7.Reverse4
10, 20, 30, Enter your choice:
1.Creation, 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.Count, 7.Reverse
```

Double Linked List Creation



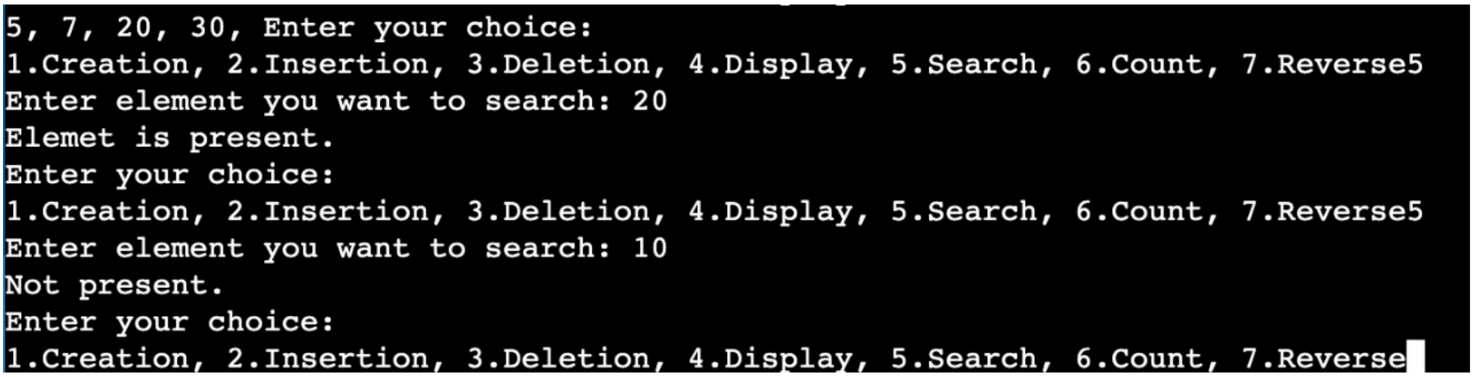
```
10, 20, 30, Enter your choice:
1.Creation, 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.Count, 7.Reverse2
Enter type of insertion:
1. Insert at first
3. Insert at given pos
1
Enter the element to insert at first:5
Enter your choice:
1.Creation, 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.Count, 7.Reverse2
Enter type of insertion:
1. Insert at first
3. Insert at given pos
3
Enter the position where you want to insert: 2
Enter the element to insert at position:7
Enter your choice:
1.Creation, 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.Count, 7.Reverse4
5, 7, 10, 20, 30, Enter your choice:
1.Creation, 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.Count, 7.Reverse
```

Double Linked List Insertion



```
5, 7, 10, 20, 30, Enter your choice:
1.Creation, 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.Count, 7.Reverse3
Enter type of deletion:
1. Delete first
2. Delete last
3. Delete at given pos
3
Enter the position of the node you want ot delete:
3
Enter your choice:
1.Creation, 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.Count, 7.Reverse4
5, 7, 20, 30, Enter your choice:
1.Creation, 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.Count, 7.Reverse3
Enter type of deletion:
1. Delete first
2. Delete last
3. Delete at given pos
2
Enter your choice:
1.Creation, 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.Count, 7.Reverse3
Enter type of deletion:
1. Delete first
2. Delete last
3. Delete at given pos
1
Enter your choice:
1.Creation, 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.Count, 7.Reverse4
7, 20, Enter your choice:
1.Creation, 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.Count, 7.Reverse
```

Double Linked List Deletion

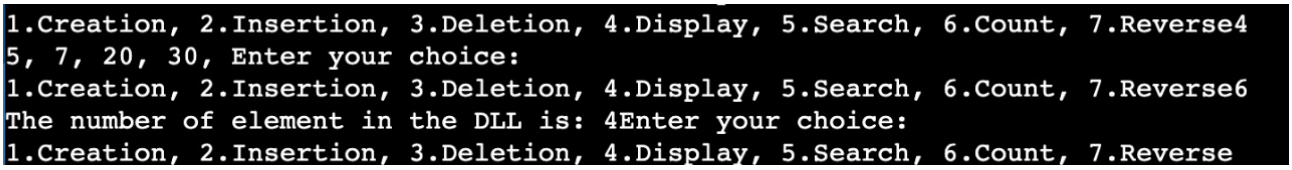


```
5, 7, 20, 30, Enter your choice:
1.Creation, 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.Count, 7.Reverse5
Enter element you want to search: 20
Elemet is present.
Enter your choice:
1.Creation, 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.Count, 7.Reverse5
Enter element you want to search: 10
Not present.
Enter your choice:
1.Creation, 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.Count, 7.Reverse
```

Search Element DLL

Count DLL

Reverse DLL



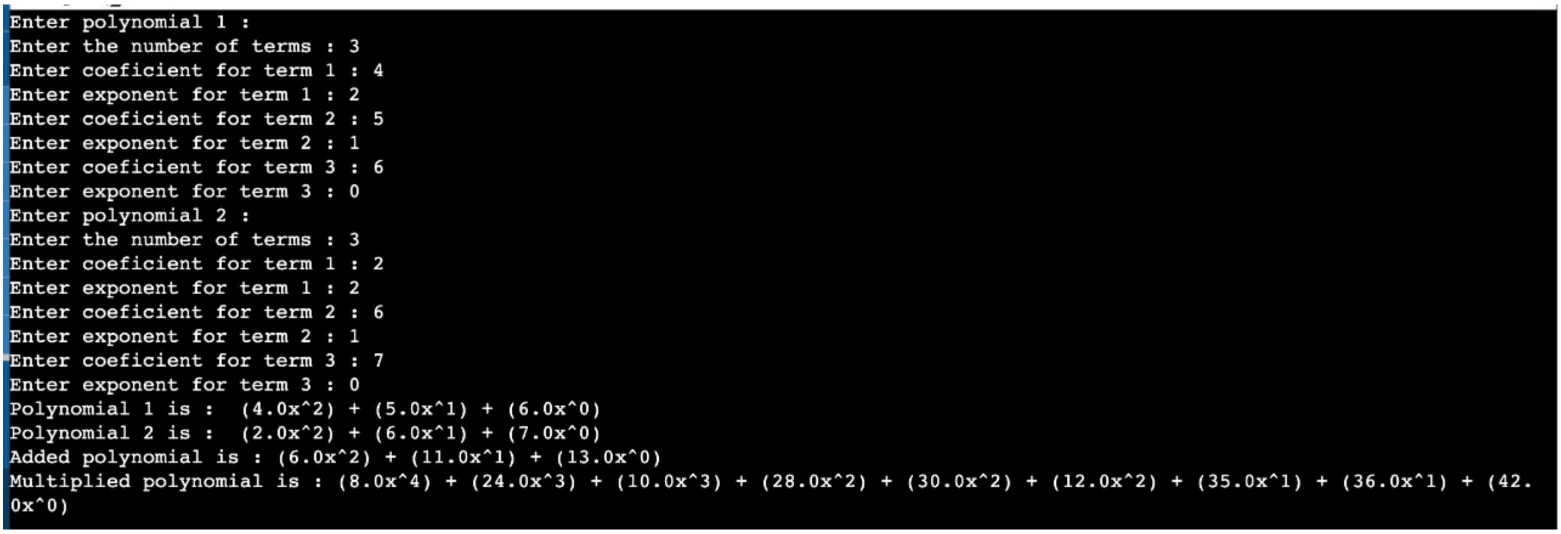
```
1.Creation, 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.Count, 7.Reverse4
5, 7, 20, 30, Enter your choice:
1.Creation, 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.Count, 7.Reverse6
The number of element in the DLL is: 4Enter your choice:
1.Creation, 2.Insertion, 3.Deletion, 4.Display, 5.Search, 6.Count, 7.Reverse
```

POLYNOMIAL ADDITION AND MULTIPLICATION

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

struct node
{
        float coef;
        int expo;
        struct node *link;
};

struct node *create(struct node *);
struct node *insert_s(struct node *,float,int);
struct node *insert(struct node *,float,int);
void display(struct node *ptr);
void poly_add(struct node *,struct node *);
void poly_mult(struct node *,struct node *);
int main( )
{
        struct node *start1=NULL,*start2=NULL;

        printf("Enter polynomial 1 :\n");
        start1=create(start1);

        printf("Enter polynomial 2 :\n");
        start2=create(start2);

        printf("Polynomial 1 is :  ");
        display(start1);
        printf("Polynomial 2 is :  ");
        display(start2);

        poly_add(start1, start2);
        poly_mult(start1, start2);
}/End of main()/

struct node *create(struct node *start)
{
        int i,n,ex;
        float co;
        printf("Enter the number of terms : ");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                printf("Enter coeficient for term %d : ",i);
                scanf("%f",&co);
                printf("Enter exponent for term %d : ",i);
                scanf("%d",&ex);
                start=insert_s(start,co,ex);
        }
        return start;
}/End of create()/
struct node *insert_s(struct node *start,float co,int ex)
{
        struct node *ptr,*tmp;
        tmp=(struct node *)malloc(sizeof(struct node));
        tmp->coef=co;
        tmp->expo=ex;
        /*list empty or exp greater than first one */
```

```
	if(start==NULL || ex > start->expo)
	{
		tmp->link=start;
		start=tmp;
	}
	else
	{
		ptr=start;
		while(ptr->link!=NULL && ptr->link->expo >= ex)
			ptr=ptr->link;
		tmp->link=ptr->link;
		ptr->link=tmp;
	}
	return start;
}/End of insert()/

struct node *insert(struct node *start,float co,int ex)
{
	struct node *ptr,*tmp;
	tmp=(struct node *)malloc(sizeof(struct node));
	tmp->coef=co;
	tmp->expo=ex;
	/If list is empty/
	if(start==NULL)
	{
		tmp->link=start;
		start=tmp;
	}
	else	/Insert at the end of the list/
	{
		ptr=start;
		while(ptr->link!=NULL)
			ptr=ptr->link;
		tmp->link=ptr->link;
		ptr->link=tmp;
	}
	return start;
}/End of insert()/

void display(struct node *ptr)
{
	if(ptr==NULL)
	{
		printf("Zero polynomial\n");
		return;
	}
	while(ptr!=NULL)
	{
		printf("(%.1fx^%d)", ptr->coef,ptr->expo);
		ptr=ptr->link;
		if(ptr!=NULL)
			printf(" + ");
		else
			printf("\n");
	}
}/End of display()/
void poly_add(struct node *p1,struct node *p2)
{
	struct node *start3;
	start3=NULL;
```

```
    while(p1!=NULL && p2!=NULL)
    {
        if(p1->expo > p2->expo)
        {
            start3=insert(start3,p1->coef,p1->expo);
            p1=p1->link;
        }
        else if(p2->expo > p1->expo)
        {
            start3=insert(start3,p2->coef,p2->expo);
            p2=p2->link;
        }
        else if(p1->expo==p2->expo)
        {
            start3=insert(start3,p1->coef+p2->coef,p1->expo);
            p1=p1->link;
            p2=p2->link;
        }
    }
    /if poly2 has finished and elements left in poly1/
    while(p1!=NULL)
    {
        start3=insert(start3,p1->coef,p1->expo);
        p1=p1->link;
    }
    /if poly1 has finished and elements left in poly2/
    while(p2!=NULL)
    {
        start3=insert(start3,p2->coef,p2->expo);
        p2=p2->link;
    }
    printf("Added polynomial is : ");
    display(start3);
}/*End of poly_add() */

void poly_mult(struct node *p1, struct node *p2)
{
    struct node *start3;
    struct node *p2_beg = p2;
    start3=NULL;
    if(p1==NULL || p2==NULL)
    {
        printf("Multiplied polynomial is zero polynomial\n");
        return;
    }
    while(p1!=NULL)
    {
        p2=p2_beg;
        while(p2!=NULL)
        {
            start3=insert_s(start3,p1->coef*p2->coef,p1->expo+p2->expo);
            p2=p2->link;
        }
        p1=p1->link;
    }
    printf("Multiplied polynomial is : ");
    display(start3);
}
```



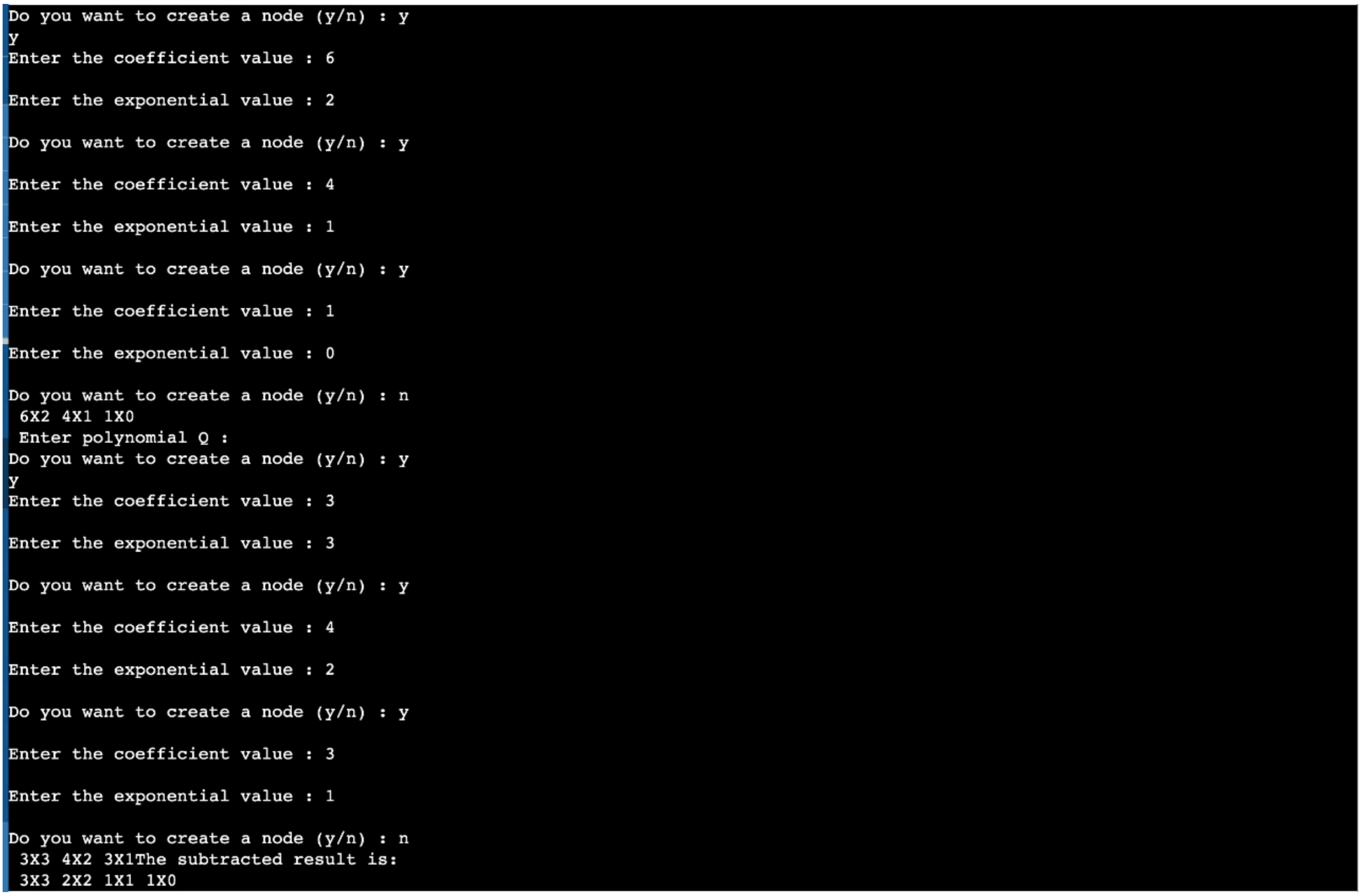
```
Enter polynomial 1 :
Enter the number of terms : 3
Enter coeficient for term 1 : 4
Enter exponent for term 1 : 2
Enter coeficient for term 2 : 5
Enter exponent for term 2 : 1
Enter coeficient for term 3 : 6
Enter exponent for term 3 : 0
Enter polynomial 2 :
Enter the number of terms : 3
Enter coeficient for term 1 : 2
Enter exponent for term 1 : 2
Enter coeficient for term 2 : 6
Enter exponent for term 2 : 1
Enter coeficient for term 3 : 7
Enter exponent for term 3 : 0
Polynomial 1 is :  (4.0x^2) + (5.0x^1) + (6.0x^0)
Polynomial 2 is :  (2.0x^2) + (6.0x^1) + (7.0x^0)
Added polynomial is : (6.0x^2) + (11.0x^1) + (13.0x^0)
Multiplied polynomial is : (8.0x^4) + (24.0x^3) + (10.0x^3) + (28.0x^2) + (30.0x^2) + (12.0x^2) + (35.0x^1) + (36.0x^1) + (42.
0x^0)
```

POLYNOMIAL SUBTRACTION

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct poly
{
      int c,e;
      struct poly *next;
};
struct poly * read_poly(struct poly *header)
{
      struct poly *p,*temp;
      char ch;
      printf("\nDo you want to create a node (y/n) : ");
      scanf("%c",&ch);
      header->c=NULL;
      header->e=NULL;
      header->next=NULL;
      printf("%c",ch);
      while(ch!='n')
      {
         p=(struct poly *)malloc(sizeof(struct poly));
         printf("\nEnter the coefficient value : ");
         scanf("%d",&p->c);
         printf("\nEnter the exponential value : ");
         scanf("%d",&p->e);
         p->next=NULL;
         if(header->next==NULL)
         {
              header->next=p;
              temp=p;
         }
         else
         {
             temp->next=p;
             temp=p;
         }
         printf("\nDo you want to create a node (y/n) : ");
         getchar();
         scanf("%c",&ch);
      }
      return header;
}

struct poly * poly_sub(struct poly *p,struct poly *q,struct poly *r)
```

```
{
    p=p->next;
    q=q->next;
    struct poly *newnode,*temp;
    r->next=NULL;
    while((p!=NULL)&&(q!=NULL))
    {
        newnode=(struct poly *)malloc(sizeof(struct poly));
        newnode->next=NULL;
        if(p->e==q->e)
        {
            newnode->c=p->c-q->c;
            newnode->e=p->e;
            p=p->next;
            q=q->next;
        }
        else if(p->e>q->e)
        {
           newnode->c=p->c;
           newnode->e=p->e;
           p=p->next;
        }
        else
        {
          newnode->c=q->c;
          newnode->e=q->e;
          q=q->next;
        }
        if(r->next==NULL)
        {
          r->next=newnode;
          temp=newnode;
        }
        else
        {
          temp->next=newnode;
          temp=newnode;
        }
    }
    if(p!=NULL)
    {
         while(p!=NULL)
         {
             newnode=(struct poly *)malloc(sizeof(struct poly));
             newnode->next=NULL;
            newnode->c=p->c;
            newnode->e=p->e;
            temp->next=newnode;
            temp=newnode;
            p=p->next;
        }
    }
    if(q!=NULL)
    {
        while(q!=NULL)
        {
            newnode=(struct poly *)malloc(sizeof(struct poly));
            newnode->next=NULL;
            newnode->c=q->c;
            newnode->e=q->e;
            temp->next=newnode;
            temp=newnode;
            q=q->next;
        }
    }
}
void traverse(struct poly *header)
{
   struct poly *ptr=header->next;
   while(ptr!=NULL)
   {
       printf(" %dX%d",ptr->c,ptr->e);
       ptr=ptr->next;
   }
}
int main()
{
   //struct poly *header=(struct poly *)malloc(sizeof(struct poly));
   //header->next=NULL;

   struct poly *p,*q,*r;
   p=(struct poly *)malloc(sizeof(struct poly));
   q=(struct poly *)malloc(sizeof(struct poly));
   r=(struct poly *)malloc(sizeof(struct poly));

      printf("\n Enter polynomial P : ");
      p=read_poly(p);
      traverse(p);
      printf("\n Enter polynomial Q : ");
      getchar();
```

```
        q=read_poly(q);
        traverse(q);
        poly_sub(p,q,r);
        printf("The subtracted result is: \n");
        traverse(r);

    getchar();
}
```



```
Do you want to create a node (y/n) : y
y
Enter the coefficient value : 6

Enter the exponential value : 2

Do you want to create a node (y/n) : y

Enter the coefficient value : 4

Enter the exponential value : 1

Do you want to create a node (y/n) : y

Enter the coefficient value : 1

Enter the exponential value : 0

Do you want to create a node (y/n) : n
 6X2 4X1 1X0
 Enter polynomial Q :
Do you want to create a node (y/n) : y
y
Enter the coefficient value : 3

Enter the exponential value : 3

Do you want to create a node (y/n) : y

Enter the coefficient value : 4

Enter the exponential value : 2

Do you want to create a node (y/n) : y

Enter the coefficient value : 3

Enter the exponential value : 1

Do you want to create a node (y/n) : n
 3X3 4X2 3X1The subtracted result is:
 3X3 2X2 1X1 1X0
```