NAME:- RASHIKA KUMARI

CLASS:- BTECH 3RD SEM

BRANCH:-IT

ROLL NO:- 2111032

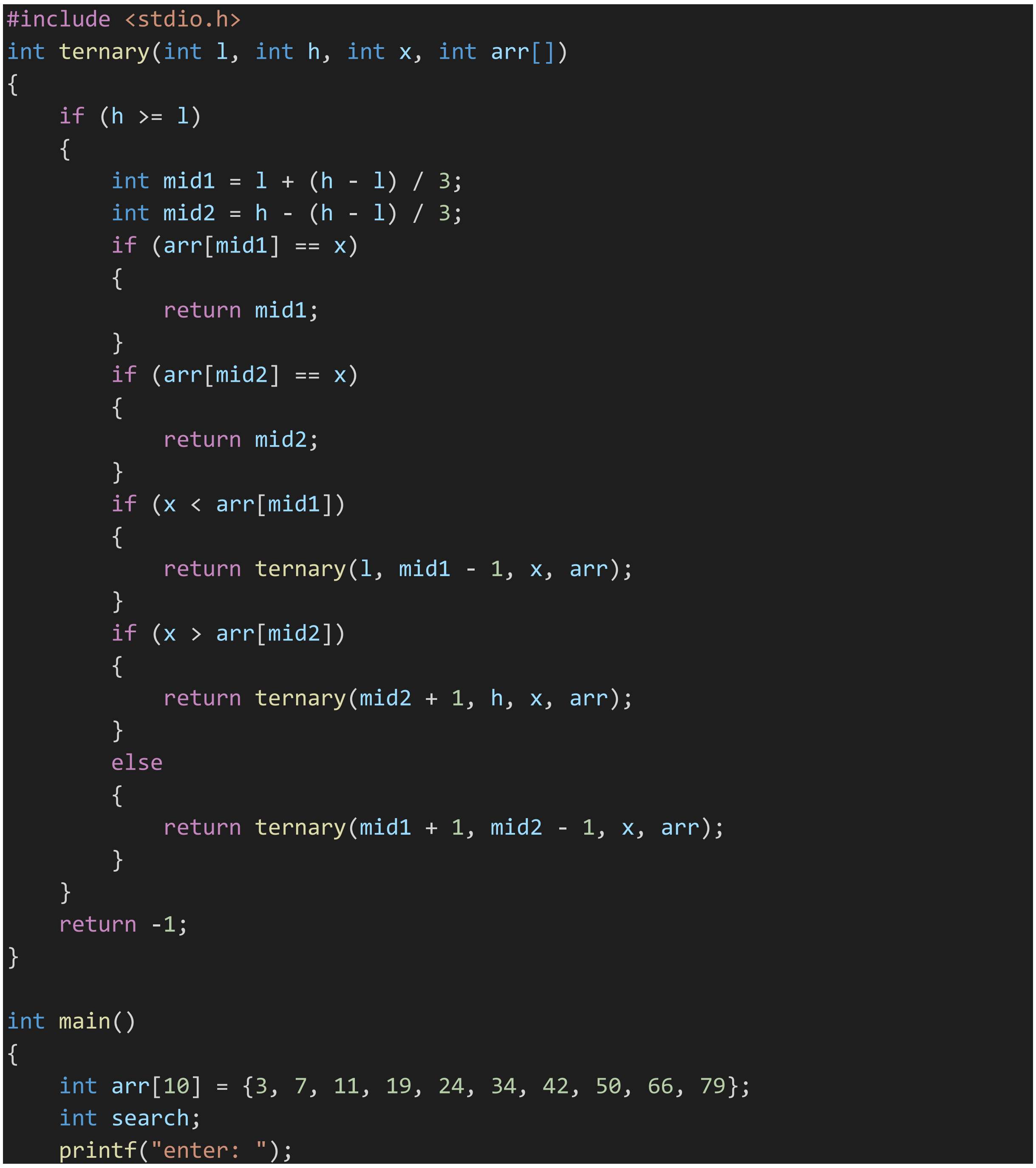
Q1. Linear Search

```c
#include <stdio.h>
int main()
{
   int arr[10] = {3, 7, 11, 19, 24, 34, 42, 50, 66, 79};
   int search, pos, flag = 0;
   printf("enter search: ");
   scanf("%d", &search);

   for (int i = 0; i < 10; i++)
   {
      if (arr[i] == search)
      {
         flag = 1;
         pos = i;
         break;
      }
   }
   if (flag == 1)
   {
      printf("Found at pos=%d", pos + 1);
   }
   else
   {
      printf("Not found");
   }
}
```

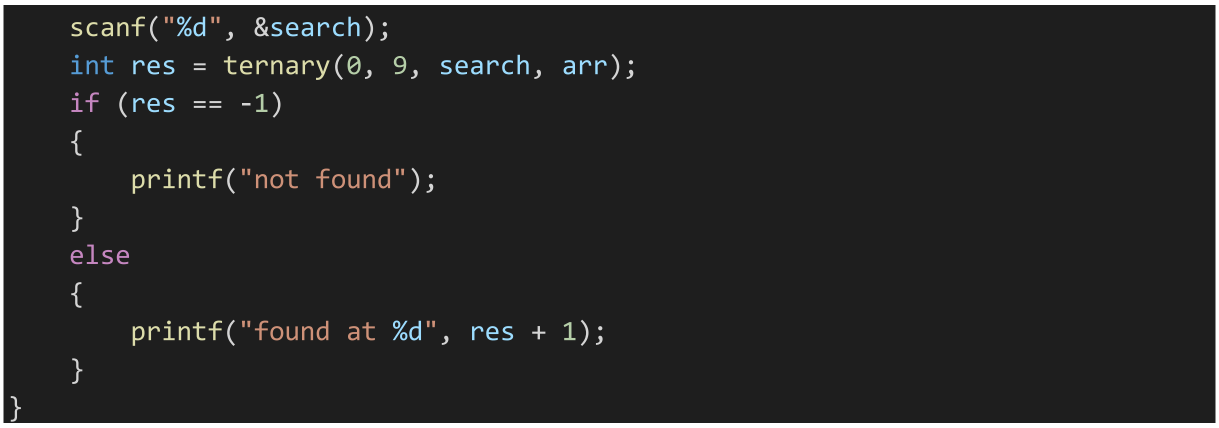
Q2. Binary Search

```c
#include <stdio.h>

int main()
{
    int arr[10] = {3, 7, 11, 19, 24, 34, 42, 50, 66, 79};
    int low, mid, high, search, pos = -1;
    low = 0, high = 9;
    printf("enter search: ");
    scanf("%d", &search);
    while (low <= high)
    {
        mid = (low + high) / 2;
        if (arr[mid] == search)
        {
            pos = mid;
            break;
        }
        else if (search < arr[mid])
        {
            high = mid - 1;
        }
        else
        {
            low = mid + 1;
        }
    }

    if (pos == -1)
    {
        printf("element not found.");
    }
    else
    {
        printf("element found at pos=%d", pos + 1);
    }
}
```

Q3. Ternary Search

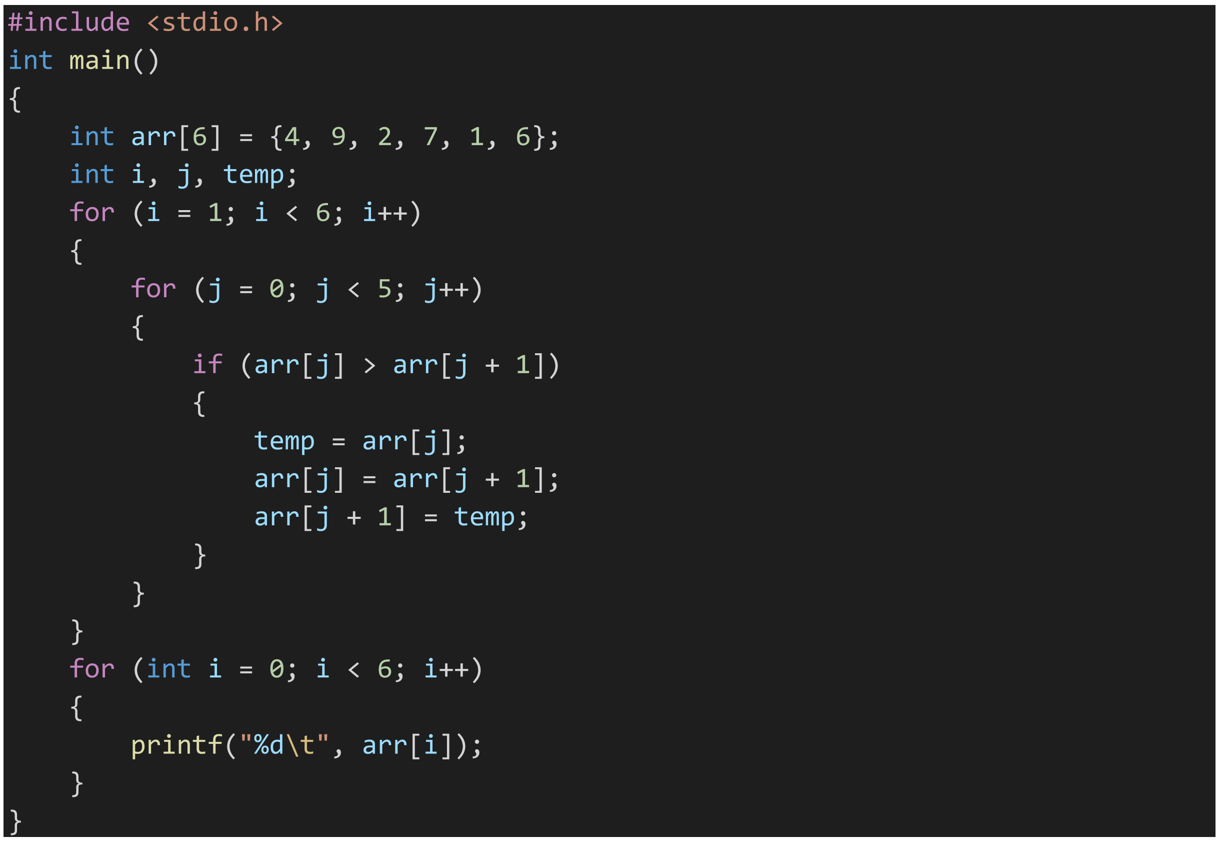


```c
#include <stdio.h>
int ternary(int l, int h, int x, int arr[])
{
    if (h >= l)
    {
        int mid1 = l + (h - l) / 3;
        int mid2 = h - (h - l) / 3;
        if (arr[mid1] == x)
        {
            return mid1;
        }
        if (arr[mid2] == x)
        {
            return mid2;
        }
        if (x < arr[mid1])
        {
            return ternary(l, mid1 - 1, x, arr);
        }
        if (x > arr[mid2])
        {
            return ternary(mid2 + 1, h, x, arr);
        }
        else
        {
            return ternary(mid1 + 1, mid2 - 1, x, arr);
        }
    }
    return -1;
}

int main()
{
    int arr[10] = {3, 7, 11, 19, 24, 34, 42, 50, 66, 79};
    int search;
    printf("enter: ");
```



```c
    scanf("%d", &search);
    int res = ternary(0, 9, search, arr);
    if (res == -1)
    {
        printf("not found");
    }
    else
    {
        printf("found at %d", res + 1);
    }
}
```

Q4. Bubble sort



```c
#include <stdio.h>
int main()
{
    int arr[6] = {4, 9, 2, 7, 1, 6};
    int i, j, temp;
    for (i = 1; i < 6; i++)
    {
        for (j = 0; j < 5; j++)
        {
            if (arr[j] > arr[j + 1])
            {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
    for (int i = 0; i < 6; i++)
    {
        printf("%d\t", arr[i]);
    }
}
```

Q5. Insertion sort

```
#include<stdio.h>
int main()
{
    int arr[10]={3,7,11,56,2,9,34,12,90,79};
    int n=10,i,j,key;

    for(j=1;j<n;j++)
    {
        key=arr[j];

        i=j-1;
        while(i>=0 && arr[i]>key)
        {
            arr[i+1]=arr[i];
            i=i-1;
        }
        arr[i+1]=key;
    }
    for(int i=0;i<n;i++)
    {
        printf("%d\t",arr[i]);
    }
}
```

Q6. Selection sort

```
#include<stdio.h>
int main()
{
    int arr[10]={3,7,11,56,2,9,34,12,90,79};
    int i,j,temp,pos;
    for(i=0;i<9;i++)
    {
        pos=i;
        for(j=i;j<10;j++)
        {
            if(arr[pos]>arr[j])
            {
                temp=arr[pos];
                arr[pos]=arr[j];
                arr[j]=temp;
            }
        }
    }
     for(int i=0;i<10;i++)
    {
        printf("%d\t",arr[i]);
    }
}
```

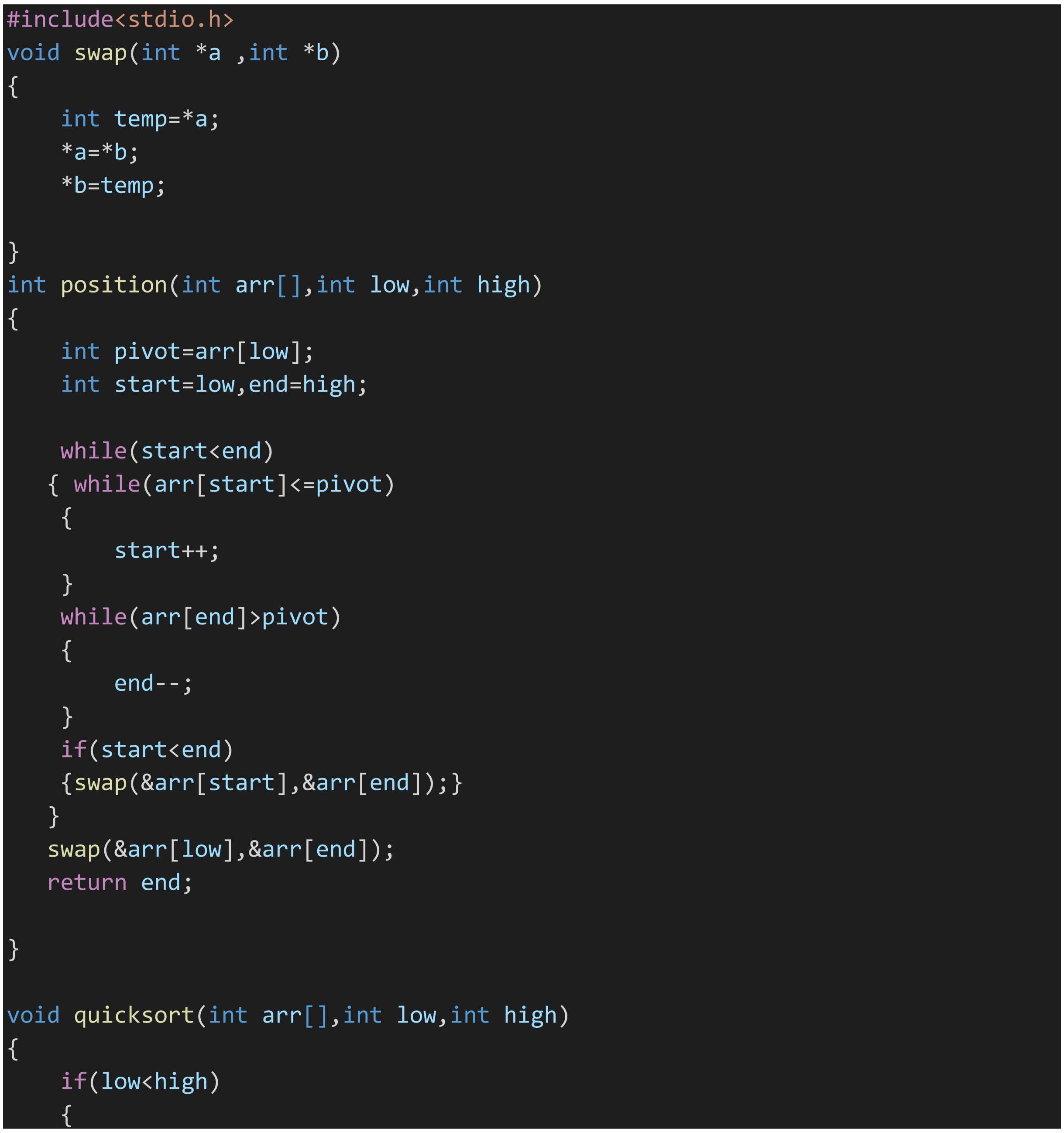
Q7. Count sort

```
#include<stdio.h>
int main()
{
    int arr[10]={3,6,2,3,4,4,1,6,8,2};
    int max=arr[0];
    for(int i=1;i<10;i++)
    {
        if(arr[i]>max)
        {
            max=arr[i];
        }
    }
    int count[max+1];
    for(int i=0;i<=max;i++)
    {
        count[i]=0;
    }
    for(int i=0;i<10;i++)
    {
        count[arr[i]]++;
    }
    for(int i=1;i<=max;i++)
    {
        count[i]+=count[i-1];
    }
    int output[10];
    for(int i=9;i>=0;i--)
    {
        output[--count[arr[i]]]=arr[i];
    }

    for(int i=0;i<10;i++)
    {
        arr[i]=output[i];
    }
    for(int i=0;i<10;i++)
    {
        printf("%d\t",arr[i]);
    }
}
```

Q8. Merge sort

```
#include<stdio.h>
void merge(int arr[],int low,int high,int mid)
{
    int i,j,k,B[high+1];
    i=low,j=mid+1,k=low;
    while(i<=mid && j<=high)
    {
        if(arr[i]<arr[j])
        {
            B[k]=arr[i];
            i++;
            k++;
        }
        else{
            B[k]=arr[j];
            k++;
            j++;
        }
    }
        while(i<=mid)
        {
            B[k]=arr[i];
            i++;
            k++;

        }
        while(j<=high)
        {
            B[k]=arr[j];
            k++;
            j++;
        }


    for(int i=low;i<=high;i++)
    {
        arr[i]=B[i];

    }
}
void mergeSort(int arr[],int low,int high)
{
    if(low<high)
    {
        int mid=(low+high)/2;
        mergeSort(arr,low,mid);
        mergeSort(arr,mid+1,high);
        merge(arr,low,high,mid);
            }
}
void display(int arr[],int n)
{
    for(int i=0;i<n;i++)
    {
        printf("%d\t",arr[i]);
    }
    printf("\n");
}
int main()
{
    int arr[8]={2,6,1,4,5,8,3,9};
    mergeSort(arr,0,7);
    display(arr,8);

}
```

Q9. Quick sort



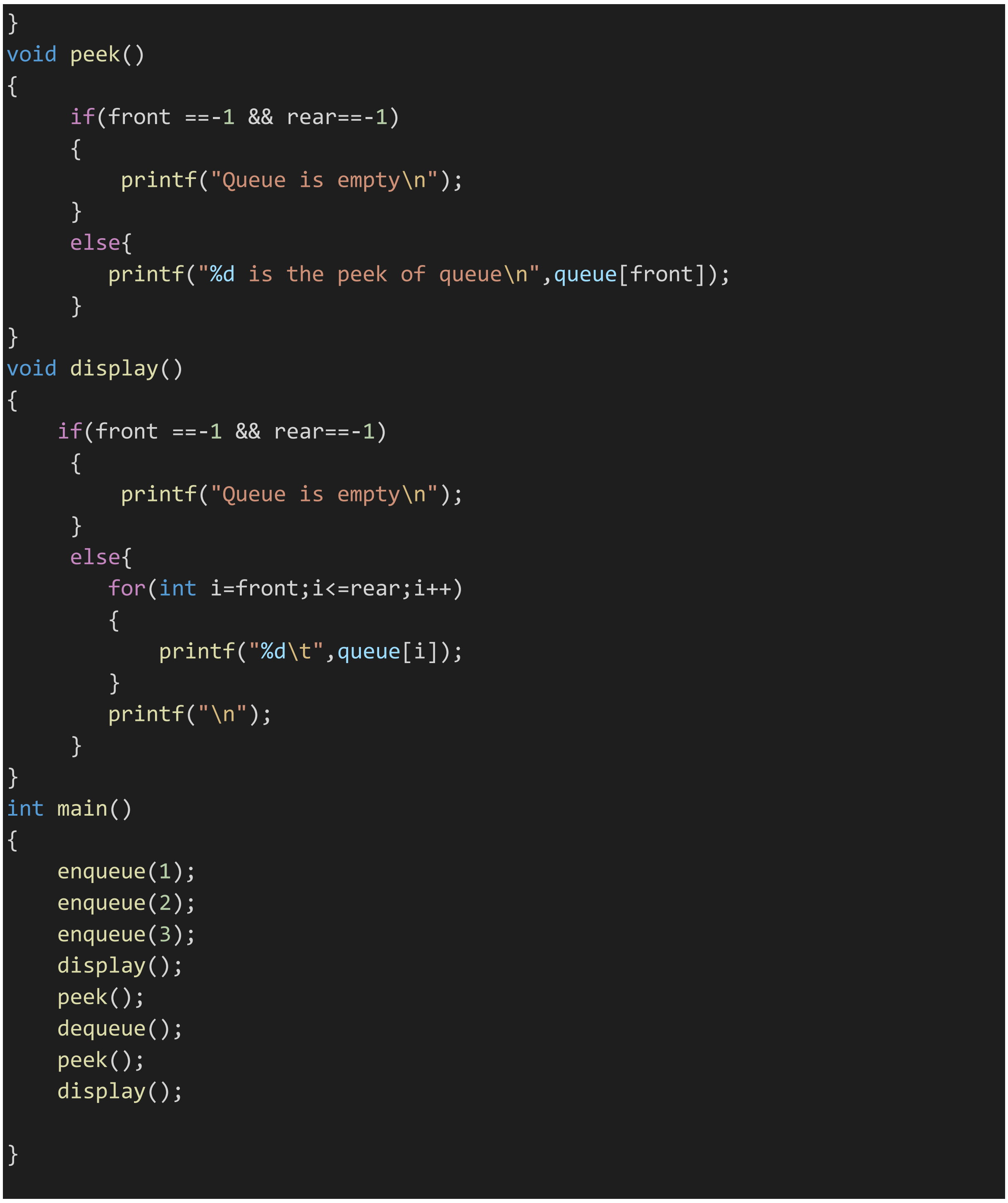
```c
#include<stdio.h>
void swap(int *a ,int *b)
{
    int temp=*a;
    *a=*b;
    *b=temp;

}
int position(int arr[],int low,int high)
{
    int pivot=arr[low];
    int start=low,end=high;

    while(start<end)
   { while(arr[start]<=pivot)
    {
        start++;
    }
    while(arr[end]>pivot)
    {
        end--;
    }
    if(start<end)
    {swap(&arr[start],&arr[end]);}
   }
   swap(&arr[low],&arr[end]);
   return end;

}

void quicksort(int arr[],int low,int high)
{
    if(low<high)
    {
        int loc=position(arr,low,high);
        quicksort(arr,low,loc-1);
        quicksort(arr,loc+1,high);
    }
}

int main()
{
    int arr[10]={35,23,89,34,86,45,19,20,65,76};
    quicksort(arr,0,9);
    for(int i=0;i<10;i++)
    printf("%d\t",arr[i]);
}
```

Q10. Stack using array

```
#include<stdio.h>
#define N 5
int stack[N];
int top=-1;
void push(int x)
{
    if(top==N-1)
    {
    printf("Stack Overflow");
    }
    else{
        top++;
        stack[top]=x;
        printf("%d is pushed into the stack\n",x);
    }

}

void pop()
{
    if(top<0)
    {
        printf("Stack underflow\n");
    }
    else{
        printf("%d is popped out\n",stack[top]);
        top--;

    }
}

void peek()
{
    if(top==-1)
    {
        printf("Stack underflow\n");
    }
    else{
        printf("%d is at the peek\n",stack[top]);
    }
}
void display()
{
    if(top==-1)
    {
        printf("stack is empty\n");
    }
    else{
        for(int i=0;i<=top;i++)
        {
            printf("%d\t",stack[i]);
        }
        printf("\n");
    }
}
int main()
{
    push(1);
    push(2);
    push(3);
    peek();
    display();
    pop();
    display();
    peek();

}
```

Q11. Queue using array

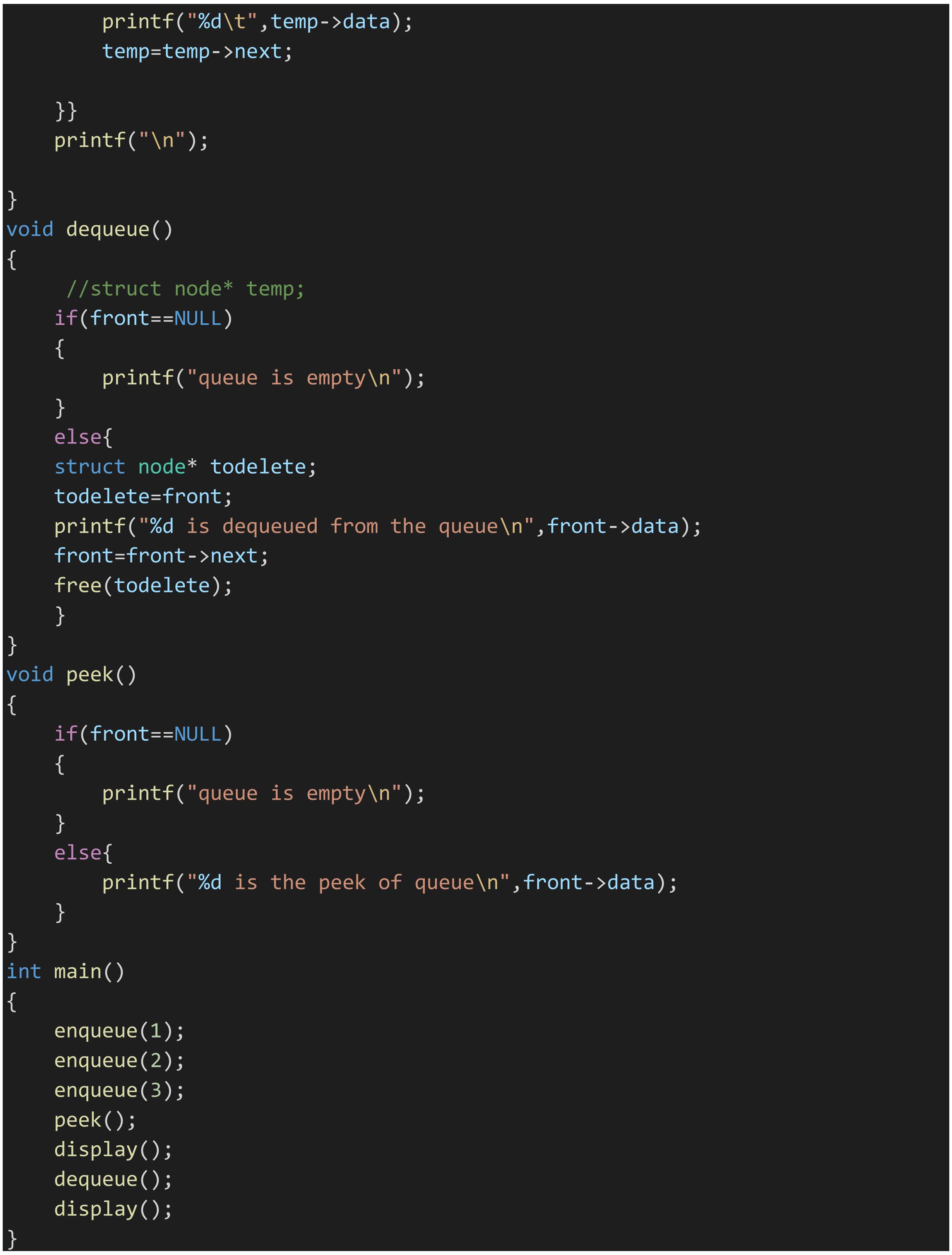
```c
#include<stdio.h>
#define N 5
int front=-1,rear=-1;
int queue[N];

void enqueue(int x)
{
    if(rear==N-1)
    {
        printf("Queue overflow\n");
    }
    else if(front==-1 && rear==-1){
        front=0;
        rear=0;
        queue[rear]=x;
        printf("%d is enqueued in the queue\n",x);
    }
    else{
        rear++;
        queue[rear]=x;
        printf("%d is enqueued in the queue\n",x);
    }
}
void dequeue()
{
    if(front ==-1 && rear==-1)
    {
        printf("Queue underflow\n");
    }
    else if(front==rear)
    {
        int x=queue[front];
        printf("%d is dequeued from queue\n",x);
        front=rear=-1;
    }
    else{
        printf("%d is dequeued from queue\n",queue[front]);
        front++;
    }
```



```c
}
void peek()
{
     if(front ==-1 && rear==-1)
     {
         printf("Queue is empty\n");
     }
     else{
        printf("%d is the peek of queue\n",queue[front]);
     }
}
void display()
{
    if(front ==-1 && rear==-1)
     {
         printf("Queue is empty\n");
     }
     else{
        for(int i=front;i<=rear;i++)
        {
            printf("%d\t",queue[i]);
        }
        printf("\n");
     }
}
int main()
{
    enqueue(1);
    enqueue(2);
    enqueue(3);
    display();
    peek();
    dequeue();
    peek();
    display();

}
```

Q12. Queue using linked list

```c
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int data;
    struct node *next;
};
struct node *front = NULL, *rear = NULL;

void enqueue(int x)
{
    struct node *newNode;
    newNode = (struct node *)malloc(sizeof(struct node));
    newNode->data = x;
    newNode->next = NULL;
    if (front == NULL && rear == NULL)
    {
        front = rear = newNode;
    }
    else
    {
        rear->next = newNode;
        rear = newNode;
    }
    printf("%d is enqueued in the queue\n",x);
}

void display()
{
    struct node* temp;
    if(front==NULL)
    {
        printf("queue is empty\n");
    }
    else{
    temp=front;

    while(temp!=NULL)
    {
```



```c
        printf("%d\t",temp->data);
        temp=temp->next;

    }}
    printf("\n");

}
void dequeue()
{
     //struct node* temp;
    if(front==NULL)
    {
        printf("queue is empty\n");
    }
    else{
    struct node* todelete;
    todelete=front;
    printf("%d is dequeued from the queue\n",front->data);
    front=front->next;
    free(todelete);
    }
}
void peek()
{
    if(front==NULL)
    {
        printf("queue is empty\n");
    }
    else{
        printf("%d is the peek of queue\n",front->data);
    }
}
int main()
{
    enqueue(1);
    enqueue(2);
    enqueue(3);
    peek();
    display();
    dequeue();
    display();
}
```

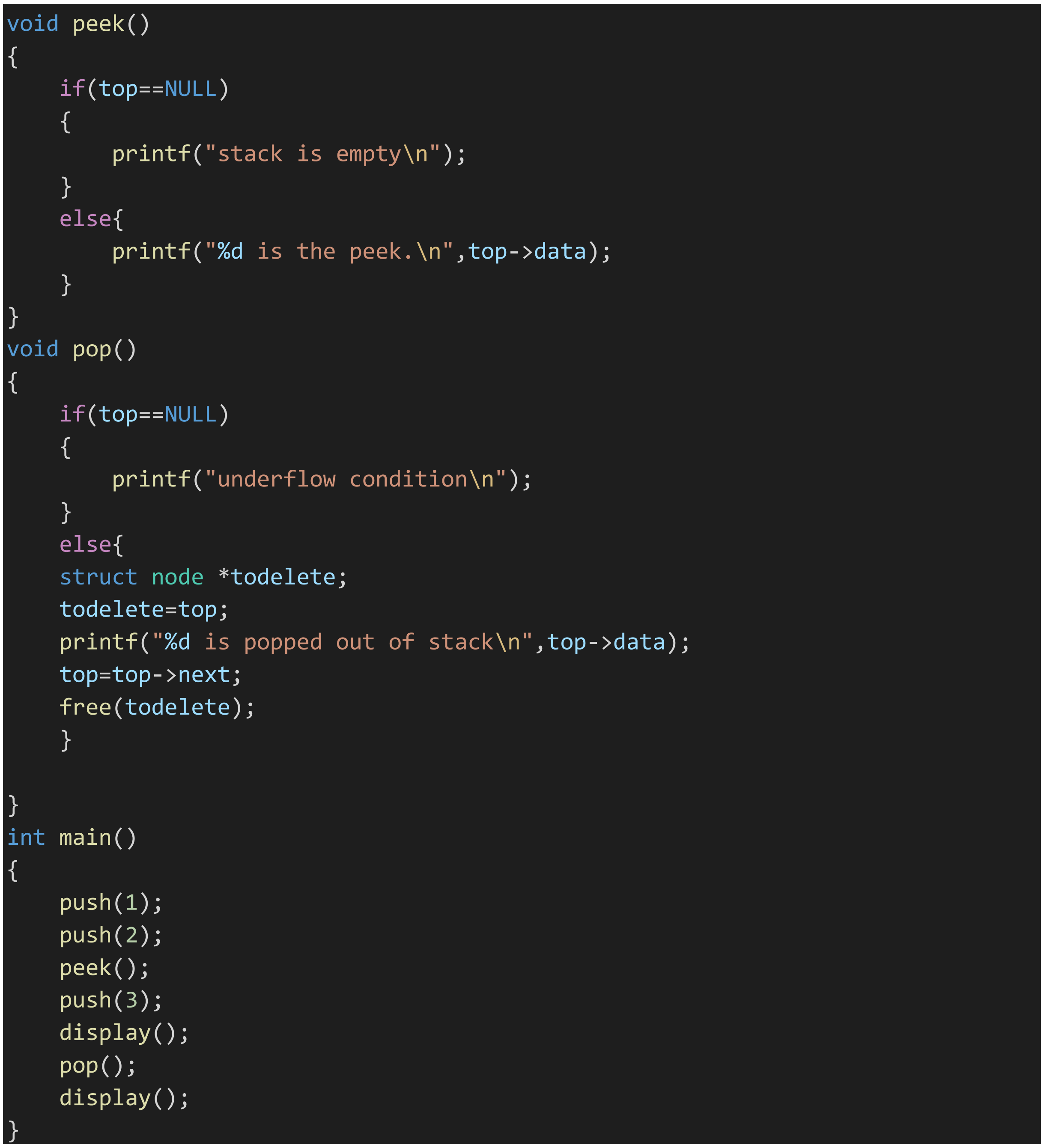
Q13. Circular Queue

```
#include <stdio.h>
#define N 3
int queue[N];
int front = -1, rear = -1;

void enqueue(int x)
{
    if (front == -1 && rear == -1)
    {
        front = rear = 0;
        queue[rear] = x;
        printf("%d is enqueued in the queue\n", queue[rear]);
    }
    else if ((rear + 1) % N == front)
    {
        printf("Queue is full");
    }
    else
    {
        rear = (rear + 1) % N;
        queue[rear] = x;
        printf("%d is enqueued in the queue\n", queue[rear]);
    }
}

void dequeue()
{
    if (front == -1 && rear == -1)
    {
        printf("Queue is empty");
    }
    else if (front == rear)
    {
        printf("%d is dequeued from the queue\n", queue[front]);
    }
    else
    {
        printf("%d is dequeued from the queue\n", queue[front]);
        front = (front + 1) % N;
    }
}
```

```c
void display()
{
    int i = front;
    if (front == -1 && rear == -1)
    {
        printf("Queue is empty\n");
    }
    else
    {
        printf("Queue is:  ");
        while (i != rear)
        {
            printf("%d\t", queue[i]);
            i = (i + 1) % N;
        }
        printf("%d", queue[rear]);
        printf("\n");
    }
}
void peek()
{
    if (front == -1 && rear == -1)
    {
        printf("Queue is empty\n");
    }
    else
    {
        printf("%d is the peek of the queue\n", queue[front]);
    }
}
int main()
{
    enqueue(1);
    enqueue(2);
    enqueue(3);

    display();

    peek();

    dequeue();
    dequeue();

    display();

    enqueue(4);
    enqueue(5);
    display();
}
```

Q 14. Stack using Linked list

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

struct node{
    int data;
    struct node *next;
};
struct node *top=NULL;
void push(int x)
{
    struct  node *newNode;
    newNode=(struct node*)malloc(sizeof(struct node));
    newNode->data=x;
    newNode->next=top;
    printf("%d is pushed into the stack\n",x);
    top=newNode;
}
void display()
{
    struct node *temp;
    temp=top;
    if(top==NULL)
    {
        printf("Underflow condition\n");
    }
    while(temp!=NULL)
    {
        printf("%d\t",temp->data);
        temp=temp->next;
    }
    printf("\n");
}
```



```c
void peek()
{
    if(top==NULL)
    {
        printf("stack is empty\n");
    }
    else{
        printf("%d is the peek.\n",top->data);
    }
}
void pop()
{
    if(top==NULL)
    {
        printf("underflow condition\n");
    }
    else{
    struct node *todelete;
    todelete=top;
    printf("%d is popped out of stack\n",top->data);
    top=top->next;
    free(todelete);
    }

}
int main()
{
    push(1);
    push(2);
    peek();
    push(3);
    display();
    pop();
    display();
}
```

Q 15. Queue using linked list

```c
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int data;
    struct node *next;
};
struct node *front = NULL, *rear = NULL;

void enqueue(int x)
{
    struct node *newNode;
    newNode = (struct node *)malloc(sizeof(struct node));
    newNode->data = x;
    newNode->next = NULL;
    if (front == NULL && rear == NULL)
    {
        front = rear = newNode;
    }
    else
    {
        rear->next = newNode;
        rear = newNode;
    }
    printf("%d is enqueued in the queue\n",x);
}

void display()
{
    struct node* temp;
    if(front==NULL)
    {
        printf("queue is empty\n");
    }
    else{
    temp=front;

    while(temp!=NULL)
    {
        printf("%d\t",temp->data);
        temp=temp->next;

    }}
    printf("\n");

}
void dequeue()
{
     //struct node* temp;
    if(front==NULL)
    {
        printf("queue is empty\n");
    }
    else{
    struct node* todelete;
    todelete=front;
    printf("%d is dequeued from the queue\n",front->data);
    front=front->next;
    free(todelete);
    }
}
void peek()
{
    if(front==NULL)
    {
        printf("queue is empty\n");
    }
    else{
        printf("%d is the peek of queue\n",front->data);
    }
}
int main()
{
    enqueue(1);
    enqueue(2);
    enqueue(3);
    peek();
    display();
    dequeue();
    display();
```

```
}
```

Q16. Insertion in singly Linked list

```
#include<stdio.h>
#include<stdlib.h>
struct Node{
    int data;
    struct Node* next;
}*head;

void createList(int n);
void traversalList();
void insertBeg(int data);
void insertAtLast(int data);
void insertAtPosition(int data,int pos);


int main()
{
    int n;
    printf("Enter the total number of nodes: ");
    scanf("%d",&n);
    createList(n);
    printf("\nData in the list\n");
    traversalList();
    insertBeg(5);
     printf("\nData in the list\n");
    traversalList();
    insertAtLast(6);
     printf("\nData in the list\n");
    traversalList();
    insertAtPosition(7,3);
     printf("\nData in the list\n");
    traversalList();

    return 0;
}

//creating list
```

```c
void createList(int n)
{
    struct Node* newNode;
    struct Node *temp;
    int data;
    head=(struct Node*)malloc(sizeof(struct Node));
    if(head==NULL)
    {
        printf("Unable to allocate memory");
        exit(0);
    }
    printf("\nEnter the data of node:");
    scanf("%d",&data);
    head->data=data;
    head->next=NULL;
    temp=head;
    for(int i=2;i<=n;i++)
    {
        newNode=(struct Node*)malloc(sizeof(struct Node));
        if(newNode==NULL)
        {
            printf("Unable to allocate memory");
            break;
        }
        printf("\nEnter the data of node:");
        scanf("%d",&data);
        newNode->data=data;
        newNode->next=NULL;
        temp->next=newNode;
        temp=temp->next;
    }
}

//traversing list

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

//insertion at beginning

void insertBeg(int data)
{
    struct Node*newNode;
    newNode=(struct Node*)malloc(sizeof(struct Node));
    if(newNode==NULL)
    {
        printf("Unable to allocate memory");
    }
    else{
        newNode->data=data;
        newNode->next=head;
        // newNode->next=head->next;
        head=newNode;
    }
}

//insertion at last

void insertAtLast(int data)
{
    struct Node*newNode, *temp;
    temp=head;
    newNode=(struct Node*)malloc(sizeof(struct Node));
     if(newNode==NULL)
    {
        printf("Unable to allocate memory");
    }
    else{
        newNode->data=data;
        newNode->next=NULL;
        while(temp->next!=NULL)
        {
            temp=temp->next;
        }
        temp->next=newNode;
    }

}

//insertion at position given

void insertAtPosition(int data,int pos)
```

```
{
     struct Node*newNode, *temp;
    temp=head;
    newNode=(struct Node*)malloc(sizeof(struct Node));
     if(newNode==NULL)
    {
        printf("Unable to allocate memory");
    }
    else{
        newNode->data=data;
        newNode->next=NULL;
        for(int i=2;i<=pos-1;i++)
        {
            temp=temp->next;
        }
        newNode->next=temp->next;
        temp->next=newNode;



    }
}
```

Q 17. Deletion in singly linked list

```
#include<stdio.h>
#include<stdlib.h>
struct Node{
    int data;
    struct Node* next;
}*head;

void createList(int n)
{
    struct Node* newNode;
    struct Node *temp;
    int data;
    head=(struct Node*)malloc(sizeof(struct Node));
    if(head==NULL)
    {
        printf("Unable to allocate memory");
        exit(0);
    }
    printf("\nEnter the data of node:");
    scanf("%d",&data);
    head->data=data;
    head->next=NULL;
    temp=head;
    for(int i=2;i<=n;i++)
    {
        newNode=(struct Node*)malloc(sizeof(struct Node));
        if(newNode==NULL)
        {
            printf("Unable to allocate memory");
            break;
        }
        printf("\nEnter the data of node:");
        scanf("%d",&data);
        newNode->data=data;
        newNode->next=NULL;
        temp->next=newNode;
        temp=temp->next;
    }
}

void deleteBeg()
{
    struct Node*toDelete;
    toDelete=head;
    head=head->next;
    free(toDelete);
}

void deleteEnd()
{
    struct Node*toDelete, *temp;

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

        temp=temp->next;
    }
    toDelete=temp->next;
    temp->next=NULL;
```

```c
    free(toDelete);

}

void traversalList()
{
    struct Node*temp;
    if(head==NULL)
    {
        printf("List is empty");
        return ;
    }
    temp=head;
    while(temp!=NULL)
    {
        printf("Data=%d\n",temp->data);
        temp=temp->next;
    }
}
void deleteAtpos(int pos)
{
    struct Node *todelete,*prev;
    todelete=prev=head;
    for(int i=2;i<=pos;i++)
    {
        prev=todelete;
        todelete=todelete->next;
    }
    prev->next=todelete->next;
    free(todelete);
}
int main()
{
int n;
    printf("Enter the total number of nodes: ");
    scanf("%d",&n);
    createList(n);
    printf("\nData in the list\n");
    traversalList();
    // deleteBeg();
    // deleteEnd();
    deleteAtpos(3);
    printf("After deletion:\n");
    traversalList();
}
```

Q18. Reverse of singly linked list

```c
#include<stdio.h>
#include<stdlib.h>
struct Node{
    int data;
    struct Node* next;
}*head;

void createList(int n)
{
    struct Node* newNode;
    struct Node *temp;
    int data;
    head=(struct Node*)malloc(sizeof(struct Node));
    if(head==NULL)
    {
        printf("Unable to allocate memory");
        exit(0);
    }
    printf("\nEnter the data of node:");
    scanf("%d",&data);
    head->data=data;
    head->next=NULL;
    temp=head;
    for(int i=2;i<=n;i++)
    {
        newNode=(struct Node*)malloc(sizeof(struct Node));
        if(newNode==NULL)
        {
            printf("Unable to allocate memory");
            break;
        }
        printf("\nEnter the data of node:");
        scanf("%d",&data);
        newNode->data=data;
        newNode->next=NULL;
        temp->next=newNode;
        temp=temp->next;
    }
}

//traversing list

void traversalList()
{
    struct Node*temp;
    if(head==NULL)
    {
        printf("List is empty");
        return ;
    }
    temp=head;
    while(temp!=NULL)
    {
        printf("Data=%d\n",temp->data);
        temp=temp->next;
    }
}
void reverse()
{
    struct Node *prevNode, *currentNode, *nextNode;
    prevNode=NULL;
    currentNode=nextNode=head;

    while(nextNode!=NULL)
    {
        nextNode=nextNode->next;
        currentNode->next=prevNode;
        prevNode=currentNode;
        currentNode=nextNode;
    }
    head=prevNode;
}
int main()
{
     int n;
    printf("Enter the total number of nodes: ");
    scanf("%d",&n);
    createList(n);
    traversalList();
    reverse();
    traversalList();
}
```

Q19. Insertion in doubly linked list

```c
#include <stdio.h>
#include <stdlib.h>
struct Node
{
    int data;
    struct Node *prev, *next;
} *head, *temp;

// creating list

void create()
{
    struct Node *newNode;
    int choice = 1;
    while (choice)
    {
        newNode = (struct Node *)malloc(sizeof(struct Node));
        printf("enter data: ");
        scanf("%d", &newNode->data);
        newNode->prev = NULL;
        newNode->next = NULL;
        if (head == NULL)
        {
            head = newNode;
            temp = newNode;
        }
        else
        {
            temp->next = newNode;
            newNode->prev = temp;
            temp = newNode;
        }
        printf("Do you want to continue? 1 or 0: ");
        scanf("%d", &choice);
    }
}

// display
void display()
{
    struct Node *temp;

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

// insertion at beginning
void insertAtBeg()
{
    struct Node *newNode;
    newNode = (struct Node *)malloc(sizeof(struct Node));
    printf("enter data: ");
    scanf("%d", &newNode->data);
    newNode->prev = NULL;
    newNode->next = head;
    head->prev = newNode;
    head = newNode;
}

// insert at last
void insertAtLast()
{
    struct Node *newNode;
    newNode = (struct Node *)malloc(sizeof(struct Node));
    printf("enter data: ");
    scanf("%d", &newNode->data);
    temp->next = newNode;
    newNode->prev = temp;
    newNode->next = NULL;
    temp = newNode;
}

// insert at position
```

```c
void insertAtPos(int pos)
{
    struct Node *newNode;
    temp = head;
    newNode = (struct Node *)malloc(sizeof(struct Node));
    printf("enter data: ");
    scanf("%d", &newNode->data);
    for (int i = 2; i <= pos - 1; i++)
    {
        temp = temp->next;
    }
    newNode->prev = temp;
    newNode->next = temp->next;
    temp->next = newNode;
    newNode->next->prev = newNode;
}

// insert after pos
void insertAfterPos(int pos)
{
    struct Node *newNode;
    temp = head;
    newNode = (struct Node *)malloc(sizeof(struct Node));
    printf("enter data: ");
    scanf("%d", &newNode->data);
    for (int i = 2; i <= pos; i++)
    {
        temp = temp->next;
    }
    newNode->prev = temp;
    newNode->next = temp->next;
    temp->next = newNode;
    newNode->next->prev = newNode;
}

int main()
{
    create();
    display();
    insertAtBeg();
    display();
    insertAtLast();
    display();
    insertAtPos(3);
    display();

}
```

Q 20. Deletion in doubly linked list

```c
#include<stdio.h>
#include<stdlib.h>
struct Node{
    int data;
    struct Node *prev, *next;
}*head,*temp;

void create(){
    struct Node *newNode;
    int choice=1;
    while(choice)
    {
    newNode=(struct Node*)malloc(sizeof(struct Node));
    printf("Enter data: ");
    scanf("%d",&newNode->data);
    newNode->next=NULL;
    newNode->prev=NULL;

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

    }
    printf("Do you want to continue? 1 or 0 :  ");
    scanf("%d",&choice);


}}
```

```
void display()
{
    temp=head;
    if(head==NULL)
    {
        printf("list is empty");
    }

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

void deleteAtFirst()
{
    struct Node *toDelete;
    toDelete=head;
    head=head->next;
    head->prev=NULL;
    free(toDelete);
}

void deleteAtLast()
{
    struct Node *toDelete;
    temp=head;
    while(temp->next!=NULL)
    {
        temp=temp->next;
    }
    toDelete=temp;
    temp->prev->next=NULL;
    free(toDelete);
}

void deleteAtPos(int pos)
{
    struct Node *toDelete;
    temp=head;
    for(int i=2;i<=pos;i++)
    {
        temp=temp->next;
    }
    toDelete=temp;
    temp->prev->next=temp->next;
    temp->next->prev=temp->prev;
    free(toDelete);

}
int main()
{
    create();
    display();
    deleteAtFirst();
    display();
    deleteAtLast();
    display();
    deleteAtPos(3);

    display();
}
```

Q21. Reverse in doubly linked list

```
#include<stdio.h>
#include<stdlib.h>
struct Node{
    int data;
    struct Node* prev,*next;
}*head,*temp;

void create()
{
    struct Node *newNode;
    int choice=1;

    while(choice)
    {newNode=(struct Node*)malloc(sizeof(struct Node));
    printf("enter data: ");
    scanf("%d",&newNode->data);
```

```c
    newNode->prev=NULL;
    newNode->next=NULL;
    if(head==NULL)
    {
        head=newNode;
        temp=newNode;
    }
    else{
        temp->next=newNode;
        newNode->prev=temp;
        temp=newNode;
    }

    printf("Do you want to continue? 1 or 0: ");
    scanf("%d",&choice);
    }
}
void display()
{
    temp=head;
    if(head==NULL)
    {
        printf("list is empty");
    }

    else{
        while(temp!=NULL)
        {
            printf("%d\t",temp->data);
            temp=temp->next;
        }
    }
    printf("\n");
}
void reverse()
{
    struct Node*nextNode,*currentNode;
    currentNode=head;

    while(currentNode!=NULL)
    {
        nextNode=currentNode->prev;
        currentNode->prev=currentNode->next;
        currentNode->next=nextNode;
        currentNode=currentNode->prev;
    }
   head=nextNode->prev;
}
```

```c
int main()
{
    create();
    display();

    printf("After reversing:\n");
    reverse();

    display();
}
```

Q 22. Insertion in circular linked list

```c
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *next;
} *head, *tail;

void create()
{
    struct node *newnode;
    head = NULL;
    int choice = 1;
    while (choice)
    {
        newnode = (struct node *)malloc(sizeof(struct node));
        printf("enter data: ");
        scanf("%d", &newnode->data);
        newnode->next = NULL;

        if (head == NULL)
        {
            head = tail = newnode;
        }
        else
        {
            tail->next = newnode;
            tail = newnode;
        }
        tail->next = head;
        printf("Do you want to continue? 1 or 0: ");
        scanf("%d", &choice);
    }
}

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

    temp = head;
    while (temp->next != head)
    {
        printf("%d\t", temp->data);
        temp = temp->next;
    }
    printf("%d\n", temp->data);

    printf("\n");
}

void insertAtBeg()
{
    struct node *newnode;
    newnode = (struct node *)malloc(sizeof(struct node));
    printf("enter data for first: ");
    scanf("%d", &newnode->data);
    if (head == NULL)
    {
        head = tail = newnode;
        tail->next = newnode;
    }
    else
    {
        newnode->next = head;
        head = newnode;
        tail->next = newnode;
```

```c
    }
}
void insertAtLast()
{
    struct node *newnode;
    newnode = (struct node *)malloc(sizeof(struct node));
    printf("enter data for last: ");
    scanf("%d", &newnode->data);
    tail->next = newnode;
    newnode->next = head;
    tail = newnode;
}

void insertAtPos(int pos)
{
    struct node *newnode, *temp;
    temp = head;
    newnode = (struct node *)malloc(sizeof(struct node));
    printf("enter data for pos %d: ",pos);
    scanf("%d", &newnode->data);

    for (int i = 2; i < pos; i++)
    {
        temp = temp->next;
    }
    newnode->next = temp->next;
    temp->next = newnode;
}

int main()
{
    create();
    display();
    insertAtBeg();
    display();
    insertAtLast();
    display();
    insertAtPos(4);
    display();
}
```

Q23. Deletion in circular linked list

```c
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *next;
} *head, *tail;

void create()
{
    struct node *newnode;
    head = NULL;
    int choice = 1;
    while (choice)
    {
        newnode = (struct node *)malloc(sizeof(struct node));
        printf("enter data: ");
        scanf("%d", &newnode->data);
        newnode->next = NULL;

        if (head == NULL)
        {
            head = tail = newnode;
        }
        else
        {
            tail->next = newnode;
            tail = newnode;
        }
        tail->next = head;
        printf("Do you want to continue? 1 or 0: ");
        scanf("%d", &choice);
    }
}
```

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

    temp = head;
    while (temp->next != head)
    {
        printf("%d\t", temp->data);
        temp = temp->next;
    }
    printf("%d\n", temp->data);

    printf("\n");
}

void deleteAtBeg()
{
    struct node *todelete;
    todelete = head;
    tail->next = head->next;
    head = head->next;
    free(todelete);
}

void deleteAtLast()
{
    struct node *todelete, *temp;
    todelete = tail->next;

    if (todelete->next == todelete)
    {
        head = NULL;
        free(head);
    }
    else
    {
        while (todelete->next != tail->next)
        {

            temp = todelete;
            todelete = todelete->next;
        }
        temp->next = tail->next;
        tail = temp;
        free(todelete);
    }
}

void deleteAtPos()
{
    int pos;
    printf("enter position: ");
    scanf("%d", &pos);
    if (pos > 4)
    {
        printf("Position doesn't contain any node! No deletion\n");
    }
    else
    {
        struct node *todelete, *temp;
        todelete = head;
        if (pos == 1)
        {
            tail->next = head->next;
            head = head->next;
            free(todelete);
        }
        else
        {
            for (int i = 2; i <= pos; i++)
            {
                temp = todelete;
                todelete = todelete->next;
            }
            temp->next = todelete->next;
            free(todelete);
        }
    }
}
int main()
{
    create();
    display();
    deleteAtBeg();
    display();
    deleteAtLast();
    display();
    deleteAtPos();
    display();
}
```

Q24. Insertion in doubly circular linked list

```c
#include <stdio.h>
#include <stdlib.h>
struct Node
{
    int data;
    struct Node *prev, *next;
} *head, *tail;

// creating list

void create()
{
    struct Node *newNode;
    int choice = 1;
    while (choice)
    {
        newNode = (struct Node *)malloc(sizeof(struct Node));
        printf("enter data: ");
        scanf("%d", &newNode->data);
        if (head == NULL)
        {
            head = tail = newNode;
            tail->next = head;
            tail->prev = head;
        }
        else
        {
            tail->next = newNode;
            newNode->prev = tail;
            newNode->next = head;
            head->prev = newNode;
            tail = newNode;
        }

        printf("Do you want to continue? 1 or 0: ");
        scanf("%d", &choice);
    }
}
```

```c
void display()
{
    struct Node *temp;
    temp = head;
    if (head == NULL)
    {
        printf("list is empty");
    }

    while (temp != tail)
    {
        printf("%d\t", temp->data);
        temp = temp->next;
    }
    printf("%d\n", temp->data);
}
void insertAtBeg()
{
    struct Node *newNode;
    newNode = (struct Node *)malloc(sizeof(struct Node));
    printf("enter data: ");
    scanf("%d", &newNode->data);

    newNode->next = head;
    head->prev = newNode;
    tail->next = newNode;
    newNode->prev = tail;
    head = newNode;
}
void insertAtLast()
{
    struct Node *newNode;
    newNode = (struct Node *)malloc(sizeof(struct Node));
    printf("enter data: ");
    scanf("%d", &newNode->data);

    tail->next = newNode;
    newNode->prev = tail;
    newNode->next = head;
    head->prev = newNode;
    tail = newNode;
}

void insertAtPos(int pos)
{
    struct Node *newNode,*temp=head;
     newNode = (struct Node *)malloc(sizeof(struct Node));
        printf("enter data: ");
        scanf("%d", &newNode->data);

        for(int i=2;i<pos;i++)
        {
            temp=temp->next;
        }
        newNode->next=temp->next;
        temp->next->prev=newNode;
        temp->next=newNode;
        newNode->prev=temp;

}

int main()
{
    create();
    display();
    insertAtBeg();
    display();
    insertAtLast();
    display();
    insertAtPos(3);
    display();
}
```

Q 25. Deletion in doubly circular linked list

```c
#include <stdio.h>
#include <stdlib.h>
struct Node
{
    int data;
    struct Node *prev, *next;
} *head, *tail;

// creating list

void create()
{
    struct Node *newNode;
    int choice = 1;
    while (choice)
    {
        newNode = (struct Node *)malloc(sizeof(struct Node));
        printf("enter data: ");
        scanf("%d", &newNode->data);
        if (head == NULL)
        {
            head = tail = newNode;
            tail->next = head;
            tail->prev = head;
        }
        else
        {
            tail->next = newNode;
            newNode->prev = tail;
            newNode->next = head;
            head->prev = newNode;
            tail = newNode;
        }

        printf("Do you want to continue? 1 or 0: ");
        scanf("%d", &choice);
    }
}

void display()
{
    struct Node *temp;
    temp = head;
    if (head == NULL)
    {
        printf("list is empty");
    }

    while (temp != tail)
    {
        printf("%d\t", temp->data);
        temp = temp->next;
    }
    printf("%d\n", temp->data);
}

void deleteAtFirst()
{
    struct Node *todelete;
    todelete=head;
    tail->next = head->next;
    head = head->next;
    head->prev = tail;
    free(todelete);

}

void deleteAtLast()
{
    struct Node *todelete,*temp;
    todelete=tail;
    tail->prev->next=head;
    head->prev=tail->prev;
    tail=tail->prev;
    free(todelete);

}

void deleteAtPos(int pos)
{
     struct Node *todelete,*temp;
     temp=head;
     for(int i=2;i<=pos;i++)
     {
        temp=temp->next;
     }
     todelete=temp;
     temp->prev->next=temp->next;
     temp->next->prev=temp->prev;
     free(todelete);

}
int main()
{
    create();
    display();
    deleteAtFirst();
    display();
    deleteAtLast();
    display();
    deleteAtPos(2);
    display();
}
```

Q26. Reverse of singly linked list using recursion

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *next;
} *head, *temp;

void create()
{
    struct node *newnode;
    int choice = 1;
    while (choice)
    {
        newnode = (struct node *)malloc(sizeof(struct node));
        printf("enter data: ");
        scanf("%d", &newnode->data);
        newnode->next = NULL;

        if (head == NULL)
        {
            head = newnode;
            temp = newnode;
        }
        else
        {
            temp->next = newnode;
            newnode->next = NULL;
            temp = newnode;
        }
        printf("Do you want to continue? 1 or 0: ");
        scanf("%d", &choice);
    }
}
struct node *reverse(struct node *temph)
{

    if (temph == NULL)
    {
        return NULL;
    }
    if(temph->next==NULL)
    {
        head=temph;
        return temph;
    }
    struct node *newhead = reverse(temph->next);
    //struct node *newhead = temph->next;
    newhead->next = temph;
    temph->next = NULL;
    return temph;
}
void traversalList()
{
    struct node *temp;
    if (head == NULL)
    {
        printf("List is empty");
        return;
    }
    temp = head;
    while (temp != NULL)
    {
        printf("%d\n", temp->data);
        temp = temp->next;
    }
}

int main()
{
    create();
    traversalList();
    struct node *newHead;
    newHead = reverse(head);
    printf("after reversing:\n");
    traversalList();
    return 0;
}
```

Q27. Reverse of doubly linked list using recursion

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
    int data;
    struct Node *prev, *next;
} *head, *temp;

// creating list

void create()
{
    struct Node *newNode;
    int choice = 1;
```

```c
    while (choice)
    {
        newNode = (struct Node *)malloc(sizeof(struct Node));
        printf("enter data: ");
        scanf("%d", &newNode->data);
        newNode->prev = NULL;
        newNode->next = NULL;
        if (head == NULL)
        {
            head = newNode;
            temp = newNode;
        }
        else
        {
            temp->next = newNode;
            newNode->prev = temp;
            temp = newNode;
        }

        printf("Do you want to continue? 1 or 0: ");
        scanf("%d", &choice);
    }
}

// display
void display()
{
    struct Node *temp;

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

struct Node *reverse(struct Node *temph)
{
    if (!temph)
    {
        return NULL;
    }
    struct Node *tempNode = temph->next;
    temph->next = temph->prev;
    temph->prev = tempNode;
    if (!temph->prev)
    {
        return temph;
    }
    return reverse(temph->prev);
}

int main()
{
    create();
    display();
    struct Node *newHead;
    head = reverse(head);
    display();
}
```

Q 28. Infix to postfix conversion

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

char stack[100];
int top=-1;

void push(char x)
{
    stack[++top]=x;

}char pop()
{
    if(top== -1)
    {
        return -1;
    }
    else{
```

```c
        return stack[top--];
    }
}

int priority(char x)
{
    if(x=='(')
    {
        return 0;
    }
    if(x=='+'|| x=='-')
    {
        return 1;
    }
    if(x=='*'||x=='/')
    {
        return 2;
    }
    if(x=='^')
    {
        return 3;
    }
    return -1;
}

int main()
{
    char exp[100];
    char *e,x;
    printf("enter the expresion: ");
    scanf("%s",exp);
    printf("\n");
    e=exp;
    while(*e!='\0')
    {
        if(isalnum(*e))
        {
            printf("%c",*e);
        }
        else if(*e=='(')
        {
            push(*e);
        }
        else if(*e==')')
        {
            while((x=pop())!='(')
            {
                printf("%c",x);
            }

        }
        else if(*e=='^' && stack[top]=='^')
        {
            push(*e);
        }
        else{
            while(priority(stack[top])>=priority(*e))
            printf("%c",pop());
            push(*e);
        }
        e++;
    }
    while(top!=-1)
    {
        printf("%c",pop());
    }
    return 0;
}
```

Q 29. Postfix Evaluation

```
#include<stdio.h>
#include<ctype.h>
#include<math.h>
int stack[20];
int top=-1;

void push(int x)
{
    stack[++top]=x;

}
int pop()
{
    if(top== -1)
    {
        return -1;
    }
    else{
        return stack[top--];
    }
}

int main()
{
   char exp[20];
   char *e;
   int num,n1,n2;
    printf("enter expression:\n");
    scanf("%s",exp);
    e=exp;
    while(*e!='\0')
    {
        if(isdigit(*e))
        {
            num=*e-48;
            push(num);


        }
        else{
            n1=pop();
            n2=pop();

            if(*e=='+')
            {
                push(n2+n1);
            }
            else if(*e=='-')
            {
                push(n2-n1);
            }
            else if(*e=='*')
            {
                push(n2*n1);
            }
            else if(*e=='/')
            {
                push(n2/n1);
            }
            else if(*e=='^')
            {
                push(pow(n2,n1));
            }
        }
        e++;
    }
    printf("result is: %d",pop());
    return 0;

}
```

Q 30. Construct Binary Tree and traverse using preorder, Postorder, Inorder

```
#include<stdio.h>
#include<stdlib.h>
struct Node{
    int data;
    struct Node *left,*right;
};

struct Node *create()
{
    int x;
    struct Node *newNode;
    newNode =(struct Node*)malloc(sizeof(struct Node));
    printf("Enter data(-1 for no node)\n");
    scanf("%d",&x);
    if (x == -1)
    {
        return NULL;
    }
    newNode->data = x;
    printf("Enter left child of %d\n",x);
    newNode->left = create();
    printf("Enter right child of %d\n",x);
    newNode->right = create();
    return newNode;
}

void Postorder(struct Node* root)
{
    if(root==NULL)
    {
        return ;
    }
    Postorder(root->left);
    Postorder(root->right);
    printf("%d\n",root->data);
```

```c
}
void Preorder(struct Node* root)
{
    if(root==NULL)
    {
        return ;
    }
    printf("%d\n",root->data);
    Postorder(root->left);
    Postorder(root->right);
}
void Inorder(struct Node* root)
{
    if(root==NULL)
    {
        return ;
    }

    Postorder(root->left);
    printf("%d\n",root->data);
     Postorder(root->right);
}

int main()
{
    struct Node *root;
    root=create();
    printf("\nPostorder:\n");
    Postorder(root);
    printf("\nPreorder:-\n");
    Preorder(root);
    printf("\nInorder:-\n");
    Inorder(root);
}
```

Q 31. Traverse using converse preorder, converse postorder, converse inorder

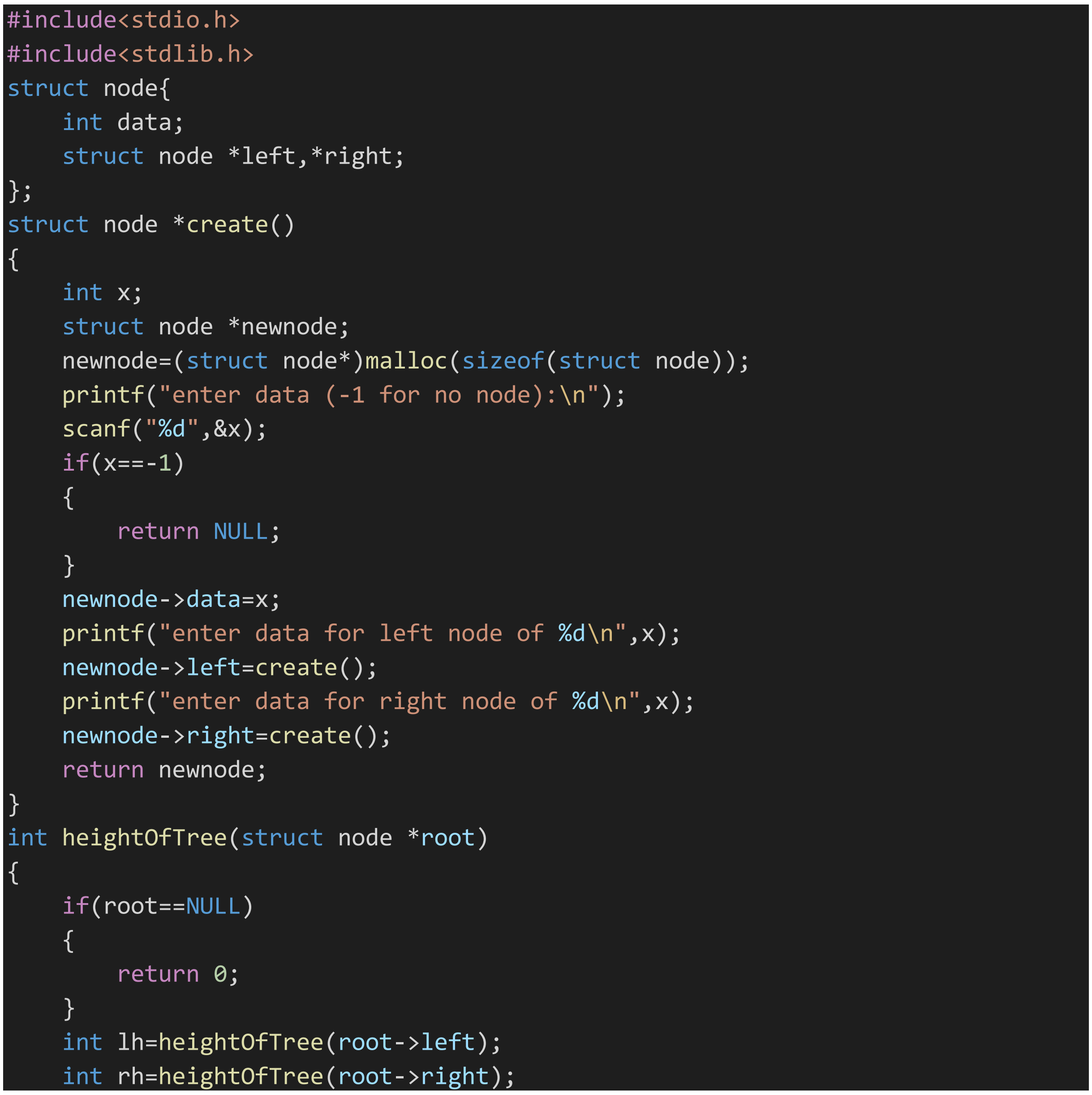
```c
#include<stdio.h>
#include<stdlib.h>
struct Node{
    int data;
    struct Node *left,*right;
};

struct Node *create()
{
    int x;
    struct Node *newNode;
    newNode =(struct Node*)malloc(sizeof(struct Node));
    printf("Enter data(-1 for no node)\n");
    scanf("%d",&x);
    if (x == -1)
    {
        return NULL;
    }
    newNode->data = x;
    printf("Enter left child of %d\n",x);
    newNode->left = create();
    printf("Enter right child of %d\n",x);
    newNode->right = create();
    return newNode;
}

void conversePostorder(struct Node* root)
{
    if(root==NULL)
    {
        return ;
    }
    conversePostorder(root->right);
    conversePostorder(root->left);
    printf("%d\n",root->data);
}
void conversePreorder(struct Node* root)
```

```
{
    if(root==NULL)
    {
        return ;
    }
    printf("%d\n",root->data);
    conversePreorder(root->right);
    conversePreorder(root->left);
}
void converseInorder(struct Node* root)
{
    if(root==NULL)
    {
        return ;
    }
    converseInorder(root->right);
    printf("%d\n",root->data);
    converseInorder(root->left);
}

int main()
{
    struct Node *root;
    root=create();
    printf("\nPostorder:\n");
    conversePostorder(root);
    printf("\nPreorder:-\n");
    conversePreorder(root);
    printf("\nInorder:-\n");
    converseInorder(root);
}
```

Q32. Total node, leaf node, non-leaf node of a binary tree

```c
#include <stdio.h>
#include <stdlib.h>
struct Node
{
    int data;
    struct Node *left, *right;
};

struct Node *create()
{
    int x;
    struct Node *newNode;
    newNode = (struct Node *)malloc(sizeof(struct Node));
    printf("Enter data(-1 for no node)\n");
    scanf("%d", &x);
    if (x == -1)
    {
        return NULL;
    }
    newNode->data = x;
    printf("Enter left child of %d\n", x);
    newNode->left = create();
    printf("Enter right child of %d\n", x);
    newNode->right = create();
    return newNode;
}

void Inorder(struct Node *root)
{

    if (root == NULL)
    {
        return;
    }
    Inorder(root->left);
    printf("%d\n", root->data);
    Inorder(root->right);
}

int countLeafNode(struct Node *root)
{
    if (root == NULL)
    {
        return 0;
    }
    if (root->right == NULL && root->left == NULL)
    {
        return 1;
    }
    else
    {
        return countLeafNode(root->left) + countLeafNode(root->right);
    }
}
int countNonLeafNode(struct Node *root){
    if(root==NULL){
        return 0;
    }
    if(root->left==NULL && root->right==NULL){
        return 0;
    }else{
        return countNonLeafNode(root->left)+countNonLeafNode(root->right) + 1;
    }
}

int countNode(struct Node *root)
{
    if (root == NULL)
    {
        return 0;
    }

        return 1 + countNode(root->left) + countNode(root->right);
}

int main()
{
    struct Node *root;
    root = create();
    printf("\nTraverse:\n");
    Inorder(root);
    int count;
    count = countNode(root);
    printf("\n%d is no of node\n", count);
    int countLeaf = countLeafNode(root);
    printf("\n%d is no of leaf node\n", countLeaf);
    int countNonLeaf=countNonLeafNode(root);
    printf("\n%d is no of non-leaf node", countNonLeaf);
}
```

Q 33. Height of a tree



```c
#include<stdio.h>
#include<stdlib.h>
struct node{
    int data;
    struct node *left,*right;
};
struct node *create()
{
    int x;
    struct node *newnode;
    newnode=(struct node*)malloc(sizeof(struct node));
    printf("enter data (-1 for no node):\n");
    scanf("%d",&x);
    if(x==-1)
    {
        return NULL;
    }
    newnode->data=x;
    printf("enter data for left node of %d\n",x);
    newnode->left=create();
    printf("enter data for right node of %d\n",x);
    newnode->right=create();
    return newnode;
}
int heightOfTree(struct node *root)
{
    if(root==NULL)
    {
        return 0;
    }
    int lh=heightOfTree(root->left);
    int rh=heightOfTree(root->right);
    if(lh>rh)
    {
        return lh+1;
    }
    else{
        return rh+1;
    }
    return 0;
}
int main()
{
    struct node *root;
    root=create();
    printf("height of tree:- %d ",heightOfTree(root));
}
```

Q34. Depth of a node in the tree

```
#include<stdio.h>
#include<stdlib.h>
struct node{
    int data;
    struct node *left,*right;
};
struct node *create()
{
    int x;
    struct node *newnode;
    newnode=(struct node*)malloc(sizeof(struct node));
    printf("enter data (-1 for no node):\n");
    scanf("%d",&x);
    if(x==-1)
    {
        return NULL;
    }
    newnode->data=x;
    printf("enter data for left node of %d\n",x);
    newnode->left=create();
    printf("enter data for right node of %d\n",x);
    newnode->right=create();
    return newnode;
}
int findDepth(struct node *root, int x)
{
    if (root == NULL)
        return -1;

    int dist = -1;

    if ((root->data == x) || (dist = findDepth(root->left, x)) >= 0 || (dist =
findDepth(root->right, x)) >= 0)
        return dist + 1;

    return dist;
}
int main()
{
    struct node *root;
    root=create();
    int n=findDepth(root,3);
    printf("depth:- %d ",n);
}
```

Q 35. Count left and right subtree

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
    int data;
    struct Node *left, *right;
};

struct Node *create()
{
```

```c
    int x;
    struct Node *newNode;
    newNode = (struct Node *)malloc(sizeof(struct Node));
    printf("Enter data(-1 for no node)\n");
    scanf("%d", &x);
    if (x == -1)
    {
        return NULL;
    }
    newNode->data = x;
    printf("Enter left child of %d\n", x);
    newNode->left = create();
    printf("Enter right child of %d\n", x);
    newNode->right = create();
    return newNode;
}

void Inorder(struct Node *root)
{

    if (root == NULL)
    {
        return;
    }
    Inorder(root->left);
    printf("%d\n", root->data);
    Inorder(root->right);
}


int leftSubtree(struct Node *root)
{
    if (root == NULL)
    {
        return 0;
    }

        return 1 + leftSubtree(root->left) + leftSubtree(root->right);
}
int rightSubtree(struct Node *root)
{
    if (root == NULL)
    {
        return 0;
    }

        return 1 + rightSubtree(root->left) + rightSubtree(root->right);
}
```

```
int main()
{
    struct Node *root;
    root = create();
    printf("\nTraverse:\n");
    Inorder(root);
    int countLeft;
    countLeft = leftSubtree(root->left);
    printf("\n%d is no of node in left Subtree\n", countLeft);
    int countRight;
    countRight = rightSubtree(root->right);
    printf("\n%d is no of node in right Subtree\n", countRight);

}
```

Q36. Construct Binary search Tree

```
#include<stdio.h>
#include<stdlib.h>
struct node{
    int data;
    struct node *left,*right;
};

struct node *createnode(int item)
{
    struct node *temp=(struct node*)malloc(sizeof(struct node*));
    temp->data=item;
    temp->right=temp->left=NULL;
    return temp;
}
```

```c
struct node *createBST(struct node *newnode,int key)
{

    if(newnode==NULL)
    {
        return createnode(key);
    }
    if (key<newnode->data)
    {
        newnode->left= createBST(newnode->left,key);
    }
    else
    {
        newnode->right= createBST(newnode->right,key);
    }
    return newnode;
}
void inorder(struct node *root)
{
    if(root!=NULL)
    {
        inorder(root->left);
        printf("%d\n",root->data);
        inorder(root->right);
    }
}
int main()
{
    struct node *root=NULL;
    root=createBST(root,10);
    root=createBST(root,14);
    root=createBST(root,7);
    root=createBST(root,8);
    root=createBST(root,6);
    root=createBST(root,3);
    root=createBST(root,4);
    root=createBST(root,1);


    printf("\nTraversal\n");
    inorder(root);
}
```

Q 37. Delete nodes in BST

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

struct node {
    int key;
    struct node *left, *right;
};


struct node* newNode(int item)
{
    struct node* temp
        = (struct node*)malloc(sizeof(struct node));
    temp->key = item;
    temp->left = temp->right = NULL;
    return temp;
}


void inorder(struct node* root)
{
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->key);
        inorder(root->right);
    }
}


struct node* insert(struct node* node, int data)
{

    if (node == NULL)
        return newNode(data);


    if (data < node->key)
        node->left = insert(node->left, data);
    else
        node->right = insert(node->right, data);


    return node;
}


struct node* minValueNode(struct node* node)
{
    struct node* current = node;


    while (current && current->left != NULL)
        current = current->left;

    return current;
}


struct node* deleteNode(struct node* root, int data)
{
    // base case
    if (root == NULL)
        return root;


    if (data < root->key)
        root->left = deleteNode(root->left, data);


    else if (data > root->key)
        root->right = deleteNode(root->right, data);


    else {

        if (root->left == NULL) {
            struct node* temp = root->right;
            free(root);
            return temp;
        }
        else if (root->right == NULL) {
            struct node* temp = root->left;
            free(root);
            return temp;
        }

```

```c
        struct node* temp = minValueNode(root->right);


        root->key = temp->key;


        root->right = deleteNode(root->right, temp->key);
    }
    return root;
}


int main()
{

    struct node* root = NULL;
    root = insert(root, 50);
    root = insert(root, 30);
    root = insert(root, 20);
    root = insert(root, 40);
    root = insert(root, 70);
    root = insert(root, 60);
    root = insert(root, 80);

    printf("Inorder traversal of the given tree \n");
    inorder(root);

    printf("\nDelete 20\n");
    root = deleteNode(root, 20);
    printf("Inorder traversal of the modified tree \n");
    inorder(root);

    printf("\nDelete 30\n");
    root = deleteNode(root, 30);
    printf("Inorder traversal of the modified tree \n");
    inorder(root);

    printf("\nDelete 50\n");
    root = deleteNode(root, 50);
    printf("Inorder traversal of the modified tree \n");
    inorder(root);

    return 0;
}
```

Q38. Creation/Deletion in AVL

```c
// AVL tree implementation in C

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

// Create Node
struct Node
{
    int key;
    struct Node *left;
    struct Node *right;
    int height;
};

int max(int a, int b);

// Calculate height
int height(struct Node *N)
{
    if (N == NULL)
        return 0;
    return N->height;
}

int max(int a, int b)
{
    return (a > b) ? a : b;
}

// Create a node
struct Node *newNode(int key)
{
    struct Node *node = (struct Node *)
        malloc(sizeof(struct Node));
    node->key = key;
    node->left = NULL;
    node->right = NULL;
    node->height = 1;
    return (node);
}

// Right rotate
struct Node *rightRotate(struct Node *y)
{
    struct Node *x = y->left;
    struct Node *T2 = x->right;

    x->right = y;
    y->left = T2;

    y->height = max(height(y->left), height(y->right)) + 1;
    x->height = max(height(x->left), height(x->right)) + 1;

    return x;
}

// Left rotate
struct Node *leftRotate(struct Node *x)
{
    struct Node *y = x->right;
    struct Node *T2 = y->left;

    y->left = x;
    x->right = T2;

    x->height = max(height(x->left), height(x->right)) + 1;
    y->height = max(height(y->left), height(y->right)) + 1;

    return y;
}

// Get the balance factor
int getBalance(struct Node *N)
{
    if (N == NULL)
        return 0;
    return height(N->left) - height(N->right);
}

// Insert node
struct Node *insertNode(struct Node *node, int key)
{
    // Find the correct position to insertNode the node and insertNode it
    if (node == NULL)
```

```
        return (newNode(key));

    if (key < node->key)
        node->left = insertNode(node->left, key);
    else if (key > node->key)
        node->right = insertNode(node->right, key);
    else
        return node;

    // Update the balance factor of each node and
    // Balance the tree
    node->height = 1 + max(height(node->left),
                           height(node->right));

    int balance = getBalance(node);

    if (balance > 1 && key < node->left->key)
        return rightRotate(node);

    if (balance < -1 && key > node->right->key)
        return leftRotate(node);

    if (balance > 1 && key > node->left->key)
    {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }

    if (balance < -1 && key < node->right->key)
    {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }

    return node;
}

struct Node *minValueNode(struct Node *node)
{
    struct Node *current = node;

    while (current->left != NULL)
        current = current->left;

    return current;
}

// Delete a nodes
```

```c
struct Node *deleteNode(struct Node *root, int key)
{
    // Find the node and delete it
    if (root == NULL)
        return root;

    if (key < root->key)
        root->left = deleteNode(root->left, key);

    else if (key > root->key)
        root->right = deleteNode(root->right, key);

    else
    {
        if ((root->left == NULL) || (root->right == NULL))
        {
            struct Node *temp = root->left ? root->left : root->right;

            if (temp == NULL)
            {
                temp = root;
                root = NULL;
            }
            else
                *root = *temp;
            free(temp);
        }

        else
        {
            struct Node *temp = minValueNode(root->right);

            root->key = temp->key;

            root->right = deleteNode(root->right, temp->key);
        }
    }

    if (root == NULL)
        return root;

    // Update the balance factor of each node and
    // balance the tree
    root->height = 1 + max(height(root->left),
                           height(root->right));

    int balance = getBalance(root);
    if (balance > 1 && getBalance(root->left) >= 0)
```

```c
        return rightRotate(root);

    if (balance > 1 && getBalance(root->left) < 0)
    {
        root->left = leftRotate(root->left);
        return rightRotate(root);
    }

    if (balance < -1 && getBalance(root->right) <= 0)
        return leftRotate(root);

    if (balance < -1 && getBalance(root->right) > 0)
    {
        root->right = rightRotate(root->right);
        return leftRotate(root);
    }

    return root;
}

// Print the tree
void printPreOrder(struct Node *root)
{
    if (root != NULL)
    {
        printf("%d ", root->key);
        printPreOrder(root->left);
        printPreOrder(root->right);
    }
}

int main()
{
    struct Node *root = NULL;

    root = insertNode(root, 2);
    root = insertNode(root, 1);
    root = insertNode(root, 7);
    root = insertNode(root, 4);
    root = insertNode(root, 5);
    root = insertNode(root, 3);
    root = insertNode(root, 8);

    printPreOrder(root);

    root = deleteNode(root, 3);

    printf("\nAfter deletion: ");
    printPreOrder(root);

    root = deleteNode(root, 5);

    printf("\nAfter deletion: ");
    printPreOrder(root);

    return 0;
}
```