INDEX

1BM22CS205

NAME: Praneeta STD.: SEC.: ROLL NO.: SUB.: DS

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1		Practice questions		
2	21/2/23	Lab program 1 (Stack)		
3	28/12/23	Lab Program 2. (Postfix)		
4	28/12/23	Postfix Evaluation Program		
5	28/12/23	Lab Program 3 - Linear queue		
6	1/1/24	Lab Program 3 - Circular queue.		
7	1/1/24	Lab Program 4 (Linked list Insertion)		
8	18/1/24	Lab Program 5 (Linked list deletion)		
9	18/1/24	Leetcode1 Minstack q		
10	18/1/24	Leetcode2 reversel.		
11	25/1/24	Lab Program 6		
12	1/2/24	Lab Program 7		
13	1/2/24	Leet code 3 (split linked list)		
14	15/2/24	Lab Program 8		
15	15/2/24	Leet code 4 (Roatate List)		
16	22/2/24	Lab Program 9.		
17	29/2/24	Lab Program 10		
18.	29/2/24	Hackarank (Pages).		

Lab program-1

3.
```
#include <stdio.h>
#define max 5
int top = -1;
int s[max];
void push (int value)
{
  if (top == max-1)
      printf("stack overflow cant push");
  else {
      top=top+1;
      s[top]=value;
    }
}
void pop()
{
    int value;
    if (top== -1)
        printf("stack is underflow cant pop\n");
    else
    {
       value =s[top];
       top=top-1;
       printf("\n%d is popped\n", value);
    }
}
void isempty()
{
    if (top== -1)
       printf("stack is empty\n");
}
void isfull()
{
    if (top==max-1)
       printf("stack is full\n"); }
```

```
void display()
{
    if (top == -1)
        printf("stack is underflow\n");
    else
    {
        printf("\n stack elements are:");
        for(int i=0; i<=top; i++)
            printf("%d\t", s[i]);
    }
}
void main()
{
    int no;
    printf("enter a no:");
    scanf("%d", &no);
    push(no);
    printf("enter a no:");
    scanf("%d", &no);
    push(no);
    printf("enter a no:");
    scanf("%d", &no);
    push(no);
    printf("enter a no:");
    scanf("%d", &no);
    push(no);
    printf("enter a no:");
    scanf("%d", &no);
    push(no);
    printf("enter a no:");
    scanf("%d", &no);
    push(no);
    display();
```

```
pop();
pop();
pop();
pop();
isempty();
isfull();
display();
pop();
pop();
}
```

Output

enter a no: 10
enter a no: 20
enter a no: 30
enter a no: 40
enter a no: 50
enter a no: 60
stack is overflow cant push
stack elements are: 10 20 30 40 50
50 is popped
40 is popped
30 is popped
20 is popped.
stack elements are: 10 10 is popped.
stack is underflow. cant pop

21/12/23

28/12/23

Week-3

~~Lab 2~~ Lab Program 2

1. // infix to postfix

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#define max 100
char s[max];
int top = -1;
void push(char value)
{
    s[++top] = value;
}
char pop()
{
    char value = ' ';
    if (top == -1)
        return value;
    else
        return s[top--];
}
int pre(char value)
{
    if (value == '^')
    {
        return 3;
    }
    else if (value == '*' || value == '/')
        return 2;
    else if (value == '+' || value == '-')
        return 1;
    else
        return 0;
}
```

```
void convert(char infix[], char postfix[])
{
    int i=0, j=0;
    char x;
    while(infix[i]!='\0')
    {
        if(infix[i]=='(')
        {
            push(infix[i]);
            i++;
        }
        else if(infix[i]==')')
        {
            while(s[top]!='(' && (top!=-1))
            {
                postfix[j++]=pop();
            }
            x=pop();
            i++;
        }
        else if(isalnum(infix[i]))
        {
            postfix[j++]=infix[i];
            i++;
        }
        else
        {
            while((s[top]!='(') && (top!=-1) && (prec[s[top]]>=preci
            {
                postfix[j++]=pop();
            }
            push(infix[i]);
            i++;
        }
    }
```

```
while ((s[top] != '(') && (top != -1))
        postfix[j++] = pop();
    postfix[j] = '\0';
}

void main()
{
    char infix[100], postfix[100];
    printf("enter the exp:");
    scanf("%s", infix);
    convert(infix, postfix);
    printf("postfix:");
    printf("%s", postfix);
}
```

O/P

enter the exp: ((a*b)+c-d)

postfix: ab*c+d-

Postfix Evalution Program

2.
```
//postfix evaluation
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define max 100
char s[max]
int top = -1;
void push(int value)
{
    if (top == max-1)
        printf("overflow, cant push");
```

```
        else
            s[++top] = value;
}
int pop()
{
    int value;
    if (top == -1)
    {
        printf("stack underflow ; cant pop\n");
        exit(1);
    }
    else {
        value = s[top--];
        return value;
    }
}
int main()
{
    char exp[100];
    char *e;
    int value = 0;
    printf("enter the exp");
    scanf("%s", exp);
    e = exp;
    while (*e != '\0')
    {
        if (isdigit(*e))
        {
            int num = *e - '0';
            push(num);
        }
        else {
```

```
char op1=pop();
char op2=pop();
switch(*e)
{
    case '+': value = op1+op2;
              break;
    case '-': value=op1*op2;
              break;
    case '*': value=op1*op2;
              break;
    case '/': value=op1/op2;
              break;
    }
    push(value);
  }
  e++;
 }
 printf("eval:%d",pop());
}
```

O/P

enter the exp: 12*34*+5-

eval: 9

Lab Program 3 linear queque

queue.

```
3 //linear queque
#include<stdio.h>
#include<stdlib.h>
#define max 4
int rear=-1;
int front=-1;
int q[max];
```

```
void enqueue(int x)
{
    if (rear == max - 1)
        printf("queue overflow\n");
    else if (front == -1 && rear == -1)
    {
        front = rear = 0;
    }
    else {
        rear++;
    }
    q[rear] = x;
}
int dequeue()
{
    int x = -1
    if (front == -1 || front > rear)
    {
        printf("\n underflow");
        return -1;
    }
    else
    {
        x = q[front];
        front++;
        if (front > rear)
            front = rear = -1;
        return x;
    }
}
void display()
{
```

```
if (front == -1 || front > rear){
        printf("\n underflow");
}
else
{
    for (int i=front; i<=rear; i++)
    {
        printf("%d \t", q[i]);
    }
}
}
void main()
{
    int c, no, x;
    while()
    {
      printf("enter 1 for insert 2 for delete 3 for display 4 for exit\n");
      printf("enter the choice:");
      scanf("%d", &c);
      switch(c)
      {
        case 1: printf("enter the no:");
                scanf("%d", &no);
                enqueue(no);
                break;
        case 2: x = dequeue();
                if (x != -1)
                {
                    printf("%d is popped \n", x);
                }
                break;
        case 3: display();
                break;
```

```
        case 4: exit(0);
        default: printf("invalid\n");
                 break;
    }
  }
}
```

Output

enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice: 1
enter the no: 10
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice: 1
enter the no: 20
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice: 1
enter the no: 30
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice: 1
10 20 30 enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice: 1
enter the no: 40
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice: 1
enter the no: 50
queue overflow
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice: 2
10 is popped
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice: 4.

11/24

Weekly ~~Lab-3~~ Lab Program - 3 Circular queue.

```
// circular queue.
#include <stdio.h>
#include <stdlib.h>
#define max 10
int rear = -1;
int front = -1;
int q[max];
int isfull()
{
    if (front == rear+1 || front == 0 && rear == max-1)
        return 1;
    return 0;
}
int is_empty()
{
    if (front == -1 && rear == -1)
        return 1;
    return 0;
}
void enqueue (int x)
{
    if (isfull())
    {
        printf ("overflow \t");
    }
    else if (front == -1 && rear == -1)
    {
        front = 0;
        rear = 0;
    }
    else {
        rear = (rear+1) % max;
    }
    q[rear] = x;
```

```
int dequeue()
{
    int value=-1;
    if (is_empty())
    {
        printf("underflow\t");
        return-1;
    }
    else
    {
        value=q[front];
        if(front==rear)
        {
            front=-1;
            rear=-1;
        }
        else{
            front=(front+1)%max;
        }
        return value;
    }
}
void display()
{
    int i;
    if(is_empty())
    {
        printf("underflow\t");
    }
    else{
        printf("elements are:");
        for(i=front;i!=rear;i=(i+1)%max)
```

```
        {
            printf("%d\t", q[c]);
        }
        printf("%d", q[i]);
    }
}
void main()
{
    int c, no, x;
    while(1)
    {
        printf("enter 1for insert 2for delete 3for display 4for exit\n");
        printf("enter the choice:");
        scanf("%d", &c);
        switch(c)
        {
            case 1: printf("enter the no:");
                    scanf("%d", &no);
                    enqueue(no);
                    break;
            case 2: x=dequeque();
                    if(x!=-1)
                        printf("%d is popped\n", x);
                    break;
            case 3: display();
                    break;
            case 4: exit(0);
            default: printf("invalid\n");
                     break;
        }
    }
}
```

Output

enter 1 for insert 2for delete 3for display 4 for exit
enter the choice : 1
enter the no:10
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice: 1
enter the no:20
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice: 3
elements are: 10 20 enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice: 2
10 is popped
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice: 2
20 is popped
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice: 2
underflow enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice: 4.

Lab Program 2 Linked list insertion

```
//Linked list
#include <stdio.h>
#include <stdlib.h>
#define max 10
int q[max];
int front = -1;
int rear = -1
struct node
{
    int data;
    struct node *next;
} * first = NULL;
void insert(struct node **p, int pos, int no)
{
```

```
struct node *t, *temp;
t = (struct node *) malloc (sizeof (struct node));
t -> data = no;
if (pos == 0)
{
    t -> next = *p;
    *p = t;
}
else
{
    struct node *temp = *p;
    for (int i = 1; i < pos && (p != NULL); i++)
        temp = temp -> next;
    t -> next = temp -> next;
    temp -> next = t;
}
}
void display (struct node *p)
{
    printf ("elements are: ");
    while (p != NULL)
    {
        printf ("%d \t", p -> data);
        p = p -> next;
    }
}
void main ()
{
    int c, n, p;
    while (1)
    {
        printf ("enter 1. insert 2. display 3. exit");
```

```
printf("enter the choice:");
scanf("%d",&c);
switch(c)
{
    case 1: printf("enter the pos and number:");
            scanf("%d%d",&p,&n);
            insert(&first,p,n);
            break;
    case 2: display(first);
            break;
    case 3: exit(0);
    default: printf("invalid input:");
             break;
  }
}
```

O/P

enter 1.insert 2.display 3.exit enter the choice: 1
enter the pos and number: 0 11
enter 1. insert 2. display 3. exit enter the choice: 1
enter the pos and number: 1 12
enter 1. insert 2. display 3. exit enter the choice: 2.
elements are: 11 12 enter 1. insert 2. display 3. exit
enter the choice: 1
enter the pos and number: 0 13
enter 1. insert 2. display 3. exit enter the choice: 2
elements are: 13 11 12 enter 1. insert 2. display 3. exit
enter the choice: 3

11/1/24

18/11/24 Week-5 Lab Program 5

1. //deletion

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

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

void delleg (struct node * a)
{
    struct node *ptr;
    if (head == NULL)
    {
        printf("list is empty");
    }
    else
    {
        ptr = head;
        head = ptr -> next;
        free (ptr);
        printf("node deleted from beginning");
    }
}

void delend (struct node * a)
{
    struct node *ptr, *ptr1;
    if (head == NULL)
    {
        printf ("list is empty");
    }
```

```
else if(head->next==NULL).
{
    free(head);
    head=NULL;
    printf("only one node is present and its deleted");
}
else
{
    ptr=head;
    while(ptr->next!=NULL)
    {
        ptr1=ptr;
        ptr=ptr->next;
    }
    ptr1->next=NULL;
    free(ptr);
    printf("element deleted at the end");
}
}
void delpos(struct node *a, int pos)
{
    if(head==NULL)
    {
        printf("List is empty");
    }
    else {
        int loc=pos;
        struct node *ptr,*ptr1;
        ptr=head;
        if(loc==1)
        {
            head=ptr->next;
            free(ptr);
```

```
        printf("Deleted at position %d\n", loc);
        return;
    }
    for(int i=0; i<loc-1; i++)
    {
        ptr1 = ptr;
        ptr = ptr->next;
        if(ptr==NULL)
        {
            printf("there are less than %d elements", loc);
            return;
        }
    }
    ptr1->next = ptr->next;
    free(ptr);
    printf("deleted at %d", loc);
  }
}
void display(struct node *s)
{
    while(s!=NULL)
    {
        printf("%d\t", s->data);
        s=s->next;
    }
}
void create(int a[], int n)
{
    struct node *last, *t;
    head = (struct node*) malloc(sizeof(struct node));
    head->data = a[0];
    head->next = NULL;
```

```
    last = head;
    for (int i=1; i<n; i++)
    {
        t = (struct node *) malloc (sizeof (struct node));
        t->data = a[i];
        t->next = NULL;
        last->next = t;
        last = t;
    }
}
void main()
{
    int a[10], n;
    printf ("enter n:");
    scanf ("%d", &n);
    printf ("enter the values");
    for (int i=0; i<n; i++)
    {
        scanf ("%d", &a[i]);
    }
    create (a, n);
    int c, l;
    while (1)
    {
        printf ("menu \n 1. delete from beg 2. del from end
            3. del at specific pos 4. display 5. exit");
        printf ("enter the choice:");
        scanf ("%d", &c);
        switch (c)
        {
            case 1: delbeg (head);
                    break;
```

```
case 2: delend(head);
        break;
case 3: printf("enter the loc");
        scanf("%d", &l);
        delpos(head, l);
        break;
case 4: display(head);
        break;
case 5: exit(0);
default: printf("invalid input");
        break;
    }
  }
}
```

Output

enter n: 5

enter the values: 1 2 3 4 5

menu

1. delete from beg 2. del from end 3. del at specific pos

4. display 5. exit enter the choice: 1

node deleted from beginning menu

1. delete from beg 2. del from end 3. del at specific pos

4. display 5. exit enter the choice: 4

2 3 4 5 menu

1: delete from beg 2. del from end 3. del at specific pos

4. display 5. exit enter the choice 2

element deleted at the end menu.

1. delete from beg 2. del from end 3. del at specific pos

4. display 5. exit enter the choice 4

2 3 4 menu

1. delete from beg 2. del from end 3. del at specific pos 4. display 5. exit enter the choice 3

enter the loc 2
deleted at 2 menu
1. delete from beg 2. del from end 3. del at specific pos
4. display 5. exit enter the choice: 5

2 leetcode) MinStack

```
typedef struct{
    int size;
    int top;
    int * s;
    int * minstack;
} MinStack;

MinStack * minStackCreate(){
    MinStack *st = (MinStack *) malloc (sizeof(MinStack));
    if (st==NULL)
    {
        printf("memory allocation failed");
        exit(0);
    }
    st->size=5;
    st->top=-1;
    st->s=(int *) malloc (st->size* sizeof(int));
    st->minstack=(int*) malloc (st->size * sizeof(int));
    if (st->s==NULL)
    {
        printf("memory allocation failed");
        free(st->s);
        free(st->minstack);
        exit(0);
    }
    return st;
}
```

```
void minStackPush (MinStack* obj, int val){
    if (obj->top == obj->size-1)
    {
        printf ("stack is overflow");
    }
    else {
        obj->top++;
        obj->s[obj->top] = val;
        if (obj->top==0 || val < obj->minstack[obj->top-1]){
            obj->minstack[obj->top] = val;
        } else {
            obj->minstack[obj->top] = obj->minstack[obj->top-1];
        }
    }
}
void minStackPop (MinStack* obj) {
    int realue;
    if (obj->top == -1)
    {
        printf ("underflow");
    }
    else
    {
        realue = obj->s[obj->top];
        obj->top--;
        printf ("%d is popped \n", realue);
    }
}
int minStackTop (MinStack * obj) {
    if (obj->top == -1)
    {
        printf ("underflow\n");
        exit (0);
    }
```

```
    else {
        value = obj->s[obj->top];
        return value;
    }
}

int minStackGetMin(MinStack* obj) {
    if (obj->top == -1)
    {
        printf("underflow\n");
        exit(0);
    }
    else {
        return obj->minstack[obj->top];
    }
}

void minStackFree(MinStack* obj) {
    free(obj->s);
    free(obj->minstack);
    free(obj);
}
```

classmate
Date
Page

Leetcode 2. reverse

```
struct ListNode* reverseBetween(struct ListNode* head, int left, int right)
{
    if(head==NULL || left==right){
        return head;
    }
    struct ListNode* p = malloc(sizeof(struct ListNode));
    p->next=head;

    struct ListNode *ptr, *ptr1;
    ptr1=p
    ptr=head
    for(int i=1;i<left;i++){
        ptr1=ptr;
        ptr=ptr->next;
    }

    struct ListNode *first=ptr;
    struct ListNode *last=ptr1;
    struct ListNode *next=NULL;

    for(int i=left;i<=right;i++){
        struct ListNode *temp=ptr->next;
        ptr->next=next;
        next=ptr;
        ptr=temp;
    }
    last->next=next;

    first->next->ptr;

    return p->next;
}
```

25/1/24

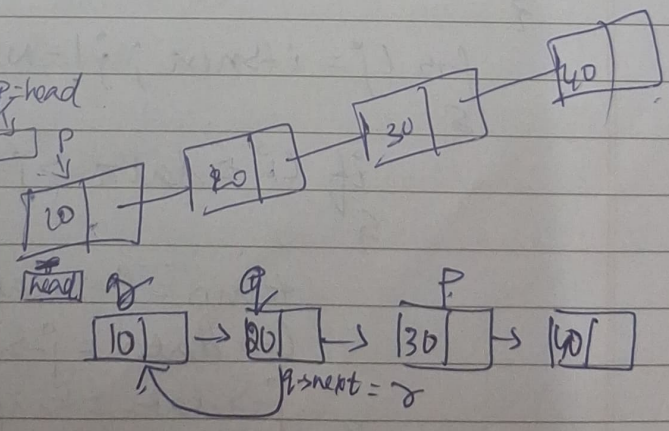
LAB - 6

Write a Program to

1.
```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *next;
}*first=NULL,*second=NULL;
void display(struct node *s)
{
    while(s!=NULL)
    {
        printf("%d\t",s->data);
        s=s->next;
    }
}
void create1(int a[],int n)
{
    struct node *last,*t;
    first=(struct node *)malloc(sizeof(struct node));
    first->data=a[0];
    first->next=NULL;
    last=first;
    for(int i=1;i<n;i++)
    {
        t=(struct node *)malloc(sizeof(struct node));
        t->data=a[i];
        t->next=NULL;
        last->next=t;
        last=t;
    }
}
```

```
void create2(int a[], int n)
{
    struct node *last, *t;
    second = (struct node *) malloc (sizeof (struct node));
    second->data = a[0];
    second->next = NULL;
    last = second;
    for(int i=1; i<n; i++)
    {
        t = (struct node*) malloc (sizeof (struct node));
        t->data = a[i];
        t->next = NULL;
        last->next = t;
        last = t;
    }
}
```

```
void reverse (struct node *p)
{
    struct node *q, *r;
    p = first;
    q = NULL;
    r = NULL;
    while (p != NULL)
    {
        r = q;
        q = p;
        p = p->next;
        q->next = r;
    }
    first = q;
}
```



```
struct node * concatat (struct node *p, struct node *q)
{
    struct node *r;
    if (p==NULL)
    {
        p=q;
        return p;
    }
    if (q==NULL)
        return q;
    r=p;
    while (r->next!=NULL)
            r=r->next;
    r->next=q;
    return p;
}
void sort (struct node *p)
{
    struct node *i, *j;
    int temp;
    for (i=p; i->next!=NULL; i=i->next)
    {
        for (j=i->next; j!=NULL; j=j->next)
        {
            if (i->data > j->data)
            {
                temp=i->data;
                i->data=j->data;
                j->data=temp;
            }
        }
    }
}
```

```
void main()
{
    int a[10], b[10], n1, n2;
    printf("enter n:");
    scanf("%d", &n1);
    printf("enter the values");
    for(int i=0; i<n1; i++)
    {
        scanf("%d", &a[i]);
    }
    struct node *s;
    s=(struct node *) malloc (sizeof (struct node));
    create1(a, n1);
    sort(first);
    printf("sorted list");
    display(first);
    reverse(first);
    printf("\n reversed list");
    display(first);
    printf("\nenter n:");
    scanf("%d", &n2);
    printf("enter the values");
    for(int i=0; i<n2; i++)
    {
        scanf("%d", &b[i]);
    }
    create2(b, n2);
    display(second);
    s=concatat(first, second);
    display(s);
}
```

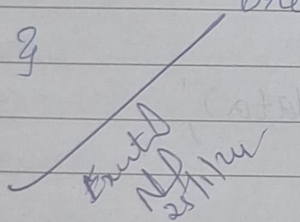
O/P enter n: 5
enter the values 9 5 7 3 1
Sorted list 1 3 5 7 9
reversed list 9 7 5 3 1
enter n: 4
enter the values 3 2 6 5
3 2 6 5
concatenated.
9 7 5 3 1 3 2 6 5

Lab Program 6b

2.
```
//Stack
#include <stdio.h>
#include <stdlib.h>

struct node
{
    int data;
    struct node *next;
}*top = NULL;
int empty(){
  if(top == NULL)
        return 1;
  return 0;
}
int full()
{
    struct node *t
    t = (struct node *) malloc(sizeof(struct node));
    if (t = NULL)
        return 1;
    return 0;
}
```

```
void push(int x)
{
    struct node *t;
    t=(struct node *) malloc (sizeof(struct node));
    if(full()){
        printf("overflow");
    }
    else{
        t->data=x;
        t->next=top;
        top=t;
    }
}
int pop(){
    struct node *t;
    t=top;
    while(t!=NULL){
        printf("%d\t", t->data);
        t=t->next;
    }
    printf("\n");
}

void main()
{
    int c, no, x;
    while(1)
    {
        printf("enter 1 for insert 2 for delete 3 for display
                4 for exit\n");
        printf("enter the choice:");
        scanf("%d", &c);
```

```
switch(c)
{
  case 1: printf("enter the no:");
          scanf("%d", &no);
          push(no);
          break;
  case 2: x = pop();
          if(x != -1){
            printf("%d is popped\n", x);
          }
          break;
  case 3: display();
          break;
  case 4: exit(0);
  default: printf("invalid\n");
           break;
}
}
}
```



O/P enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice: 1
Enter the no: 10
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice: 1
enter the no: 20
Enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice: 1
enter the no: 30
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice: 3
30 20 10

enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice: 2
30 is popped
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice: 4.

Lab program 6b

```
3 //queue.
#include <stdio.h>
#include <stdlib.h>

struct node
{
    int data;
    struct node *next;
} *front = NULL, *rear = NULL;
void enqueue(int x)
{
    struct node *t;
    t = (struct node*) malloc (sizeof(struct node));
    if (t == NULL)
    {
        printf("overflow");
    }
    t->data = x;
    t->next = NULL;
    if (front == NULL)
    {
        front = rear = t;
    }
    else {
        rear->next = t;
        rear = t;
    }
}
```

```
int dequeue()
{
    int x=-1;
    struct node *t;
    if (front==NULL)
    {
        printf("underflow");
        return x;
    }
    else {
        x=front->data;
        t=front;
        front=front->next;
        free(t);
        return x;
    }
}
void display()
{
    struct node *t;
    t=front;
    while(t!=NULL){
        printf("%d\t",t->data);
        t=t->next;
    }
    printf("\n");
}
void main()
{
    int c,no,x;
    while(1)
    {
```

```
printf("enter 1for insert 2for delete 3for display
  4 for exit \n");
printf("enter the choice:");
scanf("%d", &c);
switch(c)
{
  case 1: printf("enter the no:");
          scanf("%d", &no);
          enqueue(no);
          break;
  case 2: x = dequeue();
          if (x != -1){
            printf("%d is popped.\n", x);
          }
          break;
  case 3: display();
          break;
  case 4: exit(0);
  default: printf("invalid \n");
           break;
  }
}
}
```

o/p enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice: 1
enter the no: 10
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice: 1
enter the no: 20
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice: 1

enter the no: 30
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice: 3
10 20 30
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice: 2
10 is popped
enter 1 for insert 2 for delete 3 for display 4 for exit
enter the choice: 4

WEEK-7

Lab Program 7

```
// Doubley Linked List
#include <stdio.h>
#include <stdlib.h>
struct node
{
   int data;
   struct node *prev;
   struct node *next;
}*first = NULL;
void create (int a[], int n)
{
   struct node *t, *last;
   int i;
   first = (struct node*) malloc (sizeof (struct node));
   first -> data = a[0]
   first -> prev = first -> next = NULL;
   last = first;
   for (int i=1; i<n; i++)
   {
      t = (struct node *) malloc (sizeof (struct node));
          t -> data = a[i];
      t -> prev = last;
      t -> next = last -> next;
      last -> next = t;
      last = t;
   }
}
void display (struct node * p)
{
   while (p != NULL)
   {
```

```
        printf("%d\t", p->data);
        p=p->next;
    }
    printf("\n");
}
void insertleft(int val, int pos)
{
    struct node *t, *ptr;
    t=(struct node *) malloc(sizeof(struct node));
    int i, loc;
    loc=pos;
    if (t==NULL)
    {
        printf("overflow");
        return;
    }
    t->data=val;
    if (loc==1)
    {
        t->prev=NULL;
        t->next=first;
        if (first!=NULL)
        {
            first->prev=t;
        }
        first=t;
    }
    else {
        ptr=first;
        for(i=0; i<loc-2; i++)
        {
            ptr=ptr->next;
        }
```

```
if (ptr == NULL)
{
    printf ("cant insert ");
    return;
}
t -> next = ptr -> next;
t -> prev = ptr;
if ( ptr -> next != NULL)
{
    ptr -> next -> prev = t;
}
ptr -> next = t;
}
printf ("node inserted");
}
void deletevalue (int val)
{
    struct node *ptr;
    int value;
    value = val;
    ptr = first;

    while (ptr != NULL)
    {
        if ( ptr -> data == value)
        {
            if ( ptr -> prev != NULL)
            {
                ptr -> prev -> next = ptr -> next;
            }
            if ( ptr -> next != NULL)
            {
                ptr -> next -> prev = ptr -> prev;
            }
```

```
        if (pta == first ){
            first = pta -> next;
        }
        free (pta);
        printf ("value %d deleted", value);
        return;
        }
        pta = pta -> next;
    }
    printf ("value %d not found", value);
}

void main (){
    int a[10], n;
    int val, key, loc;
    printf ("read n");
    scanf ("%d", &n);
    printf ("enter the values:");
    for (int i=0; i<n; i++){
        scanf ("%d", &a[i]);
    }
    create (a, n);
    display (first);
    printf ("enter the value to be inserted:");
    scanf ("%d", &val);
    printf ("enter the loc to be inserted at:");
    scanf ("%d", &loc);
    insertleft (val, loc);
    display (first);
    printf ("enter the key element to be deleted");
    scanf ("%d", &key);
    deleteval (key);
    display (first);
}
```

Leetcode 3.

// Split

o/p read n : 5

enter the values : 10 20 30 40 50

10 20 30 40 50

enter the value to be inserted : 9

enter the loc to be inserted at : 2

node inserted 10 9 20 30 40 50

enter the key element to be deleted 40

value 40 deleted 10 9 20 30 50

1/2/24

Leetcode 3
//split Linked List in Parts.

```
struct ListNode** splitListToParts(struct ListNode* head,
                    int k, int* returnSize){
    struct ListNode *ptr;
    ptr = head;
    int n=0;
    while(ptr!=NULL){
        n++;
        ptr=ptr->next;
    }
    int parts = n/k;
    int extra = n%k;
    struct ListNode** a=(struct ListNode**) malloc.
        (k* sizeof(struct ListNode*));
    ptr=head;
    struct ListNode *ptr1=NULL;
    for(int i=0; i<k; i++){
        a[i]=ptr;
        int size = parts+(extra>0?1:0);
        for(int j=0; j<size; j++){
            ptr1=ptr;
            ptr=ptr->next;
        }
        if(ptr1!=NULL){
            ptr1->next=NULL;
        }
        extra--;
    }
    *returnSize=k;
    return a;
```

WEEK-8 Lab Program.8

6/8/24. // Binary tree -

```
#include <stdio.h>
#include <stdlib.h>
struct node{
    int data;
    struct node *left;
    struct node *right;
} *root = NULL;
struct node *create(struct node *t, int ele){
    if (t == NULL){
        struct node *temp = (struct node *) malloc(sizeof(struct node));
        temp->data = ele;
        temp->left = temp->right = NULL;
        return temp;
    }
    else{
        if (ele < t->data){
            t->left = create(t->left, ele);
        }
        else{
            t->right = create(t->right, ele);
        }
    }
    return t;
}

void pre(struct node *root){
    struct node *r;
    r = root;
    if (r != NULL){
        printf("%d\t", r->data);
        pre(r->left);
    } pre(r->right);
```

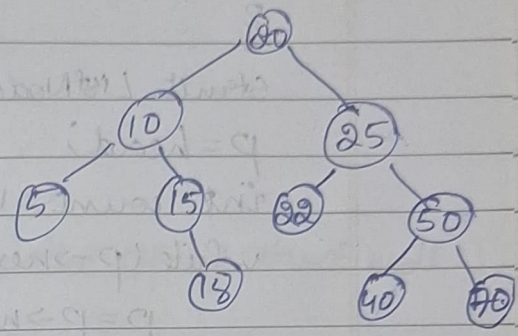
```
void in(struct node *root){
    struct node *r;
    r=root;
    if(r!=NULL){
        in(r->left);
        printf("%d\t",r->data);
        in(r->right);
    }
}

void post(struct node *root){
    struct node *r;
    r=root;
    if(r!=NULL){
        post(r->left);
        post(r->right);
        printf("%d\t",r->data);
    }
}

void main(){
    int n,ele;
    printf("enter the no of elements:");
    scanf("%d",&n);
    for(int i=0;i<n;i++){
        printf("enter the element %d:",i+1);
        scanf("%d",&ele);
        root=create(root,ele);
    }
    printf("display the elements in preorder traversal");
    pre(root);
    printf("\n display the elements in inorder traversal");
    in(root);
```

printf("\n display the elements in postorder traversal:");
post(root);

3.

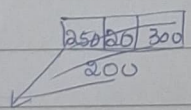
O/P

enter the no of elements: 10
enter the element 1: 20
enter the element 2: 10
enter the element 3: 5
enter the element 4: 15
enter the element 5: 25
enter the element 6: 22
enter the element 7: 50
enter the element 8: 18
enter the element 9: 40
enter the element 10: 70



display the elements in preorder traversal: 20 10 5 15 18 25 22 50 40 70
display the elements in inorder traversal: 5 10 15 18 20 22 25 40 50 70
display the elements in postorder traversal: 5 18 15 10 22 40 70 50 25 20

~~postorder~~



15/2/2024

15/2/24

Leetcode Rotate List

Question Given the head of Linked list, rotate the list to the right by k places.

```
struct ListNode* rotateRight(struct ListNode* head, int k){
    if(head==NULL || head->next==NULL || k==0){
        return head;
    }
    struct ListNode *p, *q;
    p=head;
    int count=1;
    while(p->next!=NULL){
        p=p->next;
        count++;
    }
    k=k%count;
    if(k==0){
        return head;
    }
    p->next=head;
    p=head;
    for(int i=0; i<count-k-1; i++){
        p=p->next;
    }
    q=p->next;
    p->next=NULL;
    return q;
}
```

Case 1
O/P head = [1,2,3,4,5]
k = 2
O/P = [4,5,1,2,3]

15/2/24

Case 2 head = [0,1,2]
k = 4
O/P = [2,0,1]

classmate
Date
Page

WEEK-9 Lab Program-9

29/12/24

// BFS and DFS traversal.

```
#include <stdio.h>
#include <stdlib.h>
struct node{
    int data;
    struct node *next;
}*front=NULL, *rear=NULL;
void enqueue(int x){
   struct node *t = (struct node*) malloc (sizeof (struct node));
   if (t==NULL){
        printf ("queque is overflow");
   }
   else{
        t->data=x;
        t->next=NULL;
       if (front==NULL){
            front=rear=t;
       }
       else{
            rear->next=t;
            rear=t;
       }
   }
}
int dequeque(){
      struct node *t;
      int x=-1;
      if (front==NULL){
          printf ("queque is empty");
          return x;
      }
```

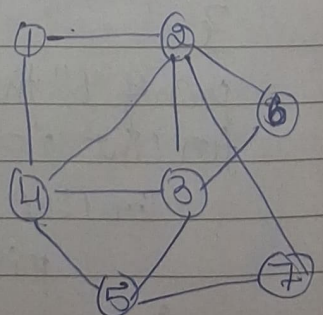
```
    else{
        t = front;
        x = t->data;
        front = front->next;
        free(t);
        return x;
    }
}

int isempt(){
    if (front == NULL){
        return 1;
    }
    return 0;
}

void bfs(int i, int visited[], int a[][20], int n){
    int u;
    printf("bfs traversal:");
    printf("%d", i);
    visited[i-1] = 1;
    enqueue(i-1);
    while(!isempt()){
        u = dequeue();
        for(int v=0; v<n; v++){
            if(a[u][v]==1 && visited[v]==0){
                printf("%d", v+1);
                visited[v] = 1;
                enqueue(v);
            }
        }
    }
}
```

```
void dfs(int i, int visited[], int a[][20], int n){
    if(visited[i-1]==0){
        printf("%d", i);
        visited[i-1]=1;
        for(int j=0; j<n; j++){
            if(a[i-1][j]==1 && visited[j]==0){
                dfs(j+1, visited, a, n);
            }
        }
    }
}

void main(){
    int visited[20]={0};
    int a[20][20];
    int n, first;
    int count=0;
    printf("enter the number of vertices:");
    scanf("%d", &n);
    printf("enter the adjacency matrix:");
    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++){
            scanf("%d", &a[i][j]);
        }
    }
    printf("the adjacency matrix:\n");
    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++)
            printf("%d\t", a[i][j]);
    }
    printf("\n");
}
```

```
printf("enter the starting vertex : ");
scanf("%d", &first);
bfs(first, visited, a, n);
for(int i=0; i<20; i++){
    visited[i]=0;
}
printf("\n dfs traversal : ");
dfs(first, visited, a, n);
for(int i=0; i<n; i++){
    if(visited[i]==0){
for(int i=0; i<n; i++){
    if(visited[i]==-1){
        count++;
    }
}
if(count==n){
    printf("\n graph is connected");
}
else{
    printf("\n graph is not connected");
}
}
```

O/P enter the number of vertices : 7
enter the adjacency matrix :

```
0 1 0 1 0 0 0
1 0 1 1 0 1 1
0 1 0 1 1 1 0
1 1 1 0 1 0 0
0 0 1 1 0 0 1
0 1 1 0 0 0 0
0 1 0 0 1 0 0
```



the adjacency matrix:

0	1	0	1	0	0	0
1	0	1	1	0	1	1
0	1	0	1	1	1	0
1	1	1	0	1	0	0
0	0	1	1	0	0	1
0	1	1	0	0	0	0
0	1	0	0	1	0	0

enter the starting vertex: 4
bfs traversal: 4 1 2 3 5 6 7
dfs traversal: 4 1 2 3 5 7 6
graph is connected

Sp.1
22/2/24

29/2/24

Lab Program 10

```
//Hash
#include <stdio.h>
#include <stdlib.h>
#define SIZE 10
struct employee{
    int key;
};
int hash(int key){
    return key % SIZE;
}
int probe(int H[], int key){
    int index = hash(key);
    int i = 0;
    while (H[(index+i) % SIZE] != 0)
        i++;
    return (index+i) % SIZE;
}
void Insert(int H[], int key){
    int index = hash(key);
    if (H[index] != 0)
        index = probe(H, key);
    H[index] = key;
    printf("Employee Id %d inserted at index %d\n",
            key, index);
}
int Search(int H[], int key){
    int index = hash(key);
    int i=0;
    while (H[(index+i) % SIZE] != key){
        if (H[(index+i) % SIZE] == 0)
            return -1;
        i++;
    }
    return (index+i) % SIZE;
```

```
int main(){
    int HT[SIZE]={0};
    Insert(HT, 1234);
    Insert(HT, 5678);
    Insert(HT, 9012);
    Insert(HT, 3456);
    Insert(HT, 3446);
    Insert(HT, 3458);
    int searchkey = 9012;
    if(resultIndex = Search(HT, searchKey);
    if(resultIndex != -1){
        printf("Employee with key %d found at index %d\n",
            searchkey, resultIndex);
    } else {
        printf("Employee with key %d not found!\n",
            searchkey);
    }
    return 0;
}
```

O/P Employee ID 1234 inserted at index 4
Employee ID 5678 inserted at index 8
Employee ID 9012 inserted at index 2
Employee ID 3456 inserted at index 7
Employee ID 3458 inserted at index 9
Employee with key 9012 found at index 21

29/2/24

```
//hackerrank question
typedef struct TreeNode {
    int data;
    struct TreeNode * left;
    struct TreeNode * right;
} TreeNode;

void inOrderTraversal (TreeNode * root, int * result, int *
  index) {
    if (root == NULL)
        return;
    inOrderTraversal (root->left, result, index);
    result [(* index)++] = root - data;
    inOrderTraversal (root->right, result, index);
}

void swapSubtrees (TreeNode* root, int k, int depth) {
    if (root == NULL)
        return;
    if (depth % k == 0) {
        TreeNode* temp = root->left;
        root->left = root->right;
        root->right = temp;
    }
    swapSubtrees (root->left, k, depth + 1);
    swapSubtrees (root->right, k, depth+1);
}

TreeNode* buildTree (int indexes rows, int indexes
    columns, int ** indexes) {
    TreeNode* root = (TreeNode*) malloc (sizeof
        (TreeNode));
```

```
root->data = 1;
root->left = NULL;
root->right = NULL;
TreeNode* nodes[indexesRows];
for(int i=1; i<indexesRows; i++){
    TreeNode* curr = nodes[i-1];

    if(indexes[i][0] != -1){
        curr->left = (TreeNode*) malloc(sizeof
            (TreeNode));
        curr->left->data = indexes[i][0];
        curr->left->right = NULL;
        curr->left->left = NULL;
        nodes[indexes[i][0]] = curr->left;
    }
    if(indexes[i][1] != -1){
        curr->right = (TreeNode*) malloc(sizeof
            (TreeNode));
        curr->right->data = indexes[i][1];
        curr->right->left = NULL;
        curr->right->right = NULL;
        nodes[indexes[i][1]] = curr->right;
    }
}
return root;
}

int** swapNodes(int indexesRows, int indexesColumns, int** indexes, int queriesCount,
int* queries, int* resultRows, int* resultColumns)
{
    int** result = (int**) malloc(queriesCount
    * sizeof(int*));
```

```
*result_rows = queriesCount;
*result_columns = indexesRows;

TreeNode* root = buildTree(indexesRows,
        indexesColumns, indexes);

for(int i=0; i<queriesCount; i++){
    int k = queries[i];
    swapSubtrees(root, k, 1);
    int* traversal = (int*) malloc(indexesRows*
                    sizeof(int));
    int index = 0;
    inOrderTraversal(root, traversal, &index);
    result[i] = traversal;
}
return result;
}
```

Test case 1

Input

3

2 3

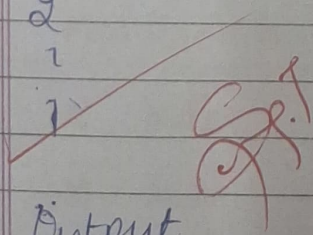
-1 -1

-1 -1

2

1

1



Output

3 1 2

2 1 3