INDEX

NAME: RANJAN DEVI STD.: _____ SEC.: _____ ROLL NO.: 1BM22CS219

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1.	21/12/23	Stack implementation using array	21/12/23	
2.	28/12/23	Infix to Postfix conversion	04/1/24	
3.	04/1/24	Queue & Circular queue	11/1/24	
4.	11/1/24	Singly linked list (Insertion) Leetcode	11/1/24	
5.	18/1/24	Singly linked list (Deletion) Leetcode	18/1/24	
6.	25/1/24	Linked list operations	25/1/24	
7.	1/2/24	Doubly linked list Stack & Queue (linked list) Leetcode	1/2/24	
8.	15/2/24	Binary Search tree Leetcode	15/2/24	
9.	22/2/24	BFS & DFS method Hackerrank	22/2/24	
10.	29/2/24	Hash Tables	29/2/24	

21/12/23

LAB 1

1. Swapping using pointers

```
#include <stdio.h>
void swap(int *, int *);
void main()
{
    int a, b;
    printf("Enter values of a and b:\n");
    scanf("%d %d", &a, &b);
    printf("Values before swapping : a=%d
            and b=%d", a, b);
    swap(&a, &b);
    printf("Values of a and b after swapping:
            a=%d and b=%d", a, b);
}

void swap(int *p, int *q)
{
    int temp;
    temp = *p;
    *p = *q;
    *q = temp;
}
```

OUTPUT:

```
Enter values of a and b:
5 4
Values before swapping : a=5 and b=4
Values of a and b after swapping : a=4 and b=5
```

2. Dynamic memory allocation

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
    int *p, *q, *s;
    int i, m, n;
    printf(" Enter no. of elements for p: ");
    scanf(" Enter no %d", &m);
    printf(" Enter no. of elements for q: ");
    scanf("%d", &n);

    p = (int *) malloc (m * sizeof(int));
    q = (int *) calloc(n, sizeof(int));

    if(p==NULL && q==NULL)
        printf("Memory is not allocated");
    else
        printf("Memory allocated successfully");
    printf("Elements of p: \n");
    for(i=0; i<m; i++)
        printf("%d \t",(i+1));

    printf("Elements of q: \n");
    for(i=0; i<n; i++)
        printf("%d \t", i);

    free(p);
    printf("\n Malloc memory successfully freed");
```

```
printf("\n Enter the new size of the
                array: ");
scanf("%d", &m);
r=(int *) realloc(q, n*sizeof(int));
if(r!=NULL)
    printf("Memory successfully reallocated
            using realloc");
else
    printf("Not allocated");
}
```

OUTPUT :-

```
Enter no. of elements for p: 5
Enter no. of elements for q: 4
Memory allocated successfully

Elements of p:
1  2  3  4  5
Elements of q:
0  1  2  3
Malloc memory successfully freed.

Enter the new size of array: 3
Memory successfully re-allocated using realloc.
```

Bafna Gold
21/12/23
Date: Page:

3. Stack Implementation.

```
#include <stdio.h>
int stack[100], i, j, ch, n, top;
void push();
void pop();
void display();
void main();
{
    int i;
    printf("Enter number of elements : ");
    scanf("%d", &n);
    while (ch != 4)
    {
        printf("Choose 1 - Push, 2 - Pop, 3 - Display
                4 - Exit \n ");
        scanf("%d", &ch);
        printf("
        switch (ch)
        {
            case 1: push();
                    break;

            case 2:
                    pop();
                    break;
            case 3:
                    display();
                    break;
            case 4:
                    printf("Exited");
                    exit(0);
                    break();
}
}
```

```
void push()
{
    int val;
    if (top == n)
        printf ("\n Overflow");
    else
    {   printf(" Enter the value: ");
        scanf ("%d", &val);
        top = top + 1;
        stack [top] = val;
    }
}

void pop()
{
    if (top == -1)
        printf ("Underflow");
    else
        top = top - 1;
}

void display()
{
    if (top == -1)
        printf ("Stack is empty");
    for (i = top; i >= 0; i--)
        printf("%d \n", stack[i]);
}
```

OUTPUT:

```
Enter number of elements: 5
Choose 1-Push, 2-Pop, 3-Display, 4-Exit:
1
Enter the value: 2
Choose 1-Push, 2-Pop, 3-Display, 4-Exit:
1
Enter the value: 3
Choose 1-Push, 2-Pop, 3-Display, 4-Exit:
1
Enter the value: 4
Choose 1-Push, 2-Pop, 3-Display, 4-Exit:
1
Enter the value: 5
Choose 1-Push, 2-Pop, 3-Display, 4-Exit:
3
Stack is: 2 3 4 5
Choose 1-Push, 2-Pop, 3-Display, 4-Exit:
2
Choose 1-Push, 2-Pop, 3-Display, 4-Exit:
3
Stack is: 2 3 4
Choose 1-Push, 2-Pop, 3-Display, 4-Exit:
4
Exited.
```

21/12/2023

"%[^\n]"

strcpy(str[top], "#");

28/12/24

LAB - 9

1. Write a program to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators.

```
#include <stdio.h>
#include <string.h>
int i, j=0, n, top=-1;
char val, temp, c;
char stack[100];
char infix[100], postfix[100];
void push(char);
char pop();
int precedence(char);

void main()
{
    printf("Enter an infix expression : \n");
    scanf("%s", infix);
    n=strlen(infix);
    for(i=0; i<=n; i++)
    {
        if(infix[i]=='(')
            push(infix[i]);
        else if(infix[i]>='a' && infix[i]<='z' ||
                infix[i]>=0 && infix[i]<=9 ||
                infix[i]>='A' && infix[i]<='Z')
        {
            postfix[j]=infix[i];
        } j++;
```

```
else if (infix[i] == ')')
{
    while (stack[top] != '(')
    {
        postfix[j] = pop();
        j++;
    }
    temp = pop();
}
else
{
    while (precedence(infix[i]) <=
        precedence(stack[top]) && top>=0)
    {
        postfix[j] = pop();
        j++;
    }
    push(infix[i]);
}
}
while (stack[top] != '\0')
{
    postfix[j] = pop();
    j++;
}
postfix[j] = '\0';
printf("Postfix expression :\n");
printf("%s", postfix);
}
```

```
void push(char val)
{
    if(top == n-1)
        printf("Overflow");
    else
    {   top += 1;
        stack[top] = val;
    }
}
char pop()
{
    if(top == -1)
        printf("Underflow");
    else
    {   val = stack[top];
        top -= 1;
        return val;
    }
}
int precedence(char c)
{   if(c == '^')
        return 5;
    elseif(c == '/')
        return 4;
    else if(c == '*')
        return 3;
    else if(c == '+')
        return 2;
    else if(c == '-')
        return 1;
    else
        return -1;
}
```

OUTPUT:

```
Enter an infix expression :
a*b+c*d
Postfix expression :
ab* cd* +
```

2. Evaluation of postfix expression

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
int n, top=-1, k;
int val, res, ans;
int stack[100];
char expn[100];
void push(int[], int);
int pop(int[]);
int evaluate(char expn[]);

void main()
{
    printf("Enter a postfix expression:\n");
    scanf("%s", expn);
    ans = evaluate(expn);
    printf("Answer of above expression : \n");
    printf("%d", ans);
}
```

```
int evaluate (char expn[])
{
    int op1, op2, i=0;
    n= strlen(expn);
    while (expn[i] != '\0')
    {
        if(isdigit (expn[i]))
            push (stack, (int) (expn[i]-'0'));
        else
        {
            op1 = pop (stack);
            op2 = pop (stack);
            switch (expn[i])
            {
                case '+' :
                    res = op1 + op2;
                    break;
                case '-' :
                    res=op1-op2;
                    break;
                case '*' :
                    res = op1 * op2;
                    break;
                case '/' :
                    res = op2 / op1;
                    break;
                case '%' :
                    res = op2 % op1;
                    break;
                default :
                    continue;
            }
```

```
            if(res<0)
                res = res*(-1);
            push(stack, res);
        }
        i++;
    }
    return stack[top];
}

void push (int stack[], int val)
{
    if(top==n-1)
        printf("Overflow");
    else
    {   top+=1;
        stack[top] = val;
    }
}

int pop(int stack[])
{   if(top==-1)
        printf("Underflow");
    else
    {   val = stack[top];
        top-=1;
        return val;
    }
}
```

OUTPUT:

Enter a postfix expression:
12*34*+5-
Value : 9

WEEK 3

1. Queue Implementation

```
#include <stdio.h>
int q[50], rear = -1, front = -1, size;
void enqueue();
void dequeue();
void display();
void main()
{
    int ch;
    printf("Enter the size of queue :");
    scanf("%d", &size);
    printf("Enter choice :");
    printf("Press 1. insert, 2. delete, 3. Display
            and 4. Exit \n");
    while(ch != 4)
    {
        printf("Enter choice :");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1 :
                enqueue();
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
    } }
```

```
        printf("Exited");
}

void enqueue()
{
    int item;
    if(rear == size-1)
        printf("Queue is full\n");
    else
    {   if(front == -1)
            front = 0;
        printf("Insert an element: ");
        scanf("%d", &item);
        rear += 1;
        q[rear] = item;
    }
}

void dequeue()
{
    if (front == -1 || front > rear)
        printf("Queue is empty\n");
    else
    {
        printf("Deleted element is: %d\n", q[front]);
        front += 1;
    }
}

void display()
{
    int i;
```

```
    if (front == -1)
        printf("Queue is empty");
    else
    {
        printf("Queue is: \n");
        for(int i=front; i<=rear; i++)
            printf("%d\t", q[i]);
        printf("\n");
    }
}
```

OUTPUT -

```
Press - 1.Insert, 2.delete, 3.Display and 4.Exit
Enter choice: 2
Queue is empty
Enter choice: 1
Insert an element: 6
Enter choice: 1
Insert an element: 8
Enter choice: 1
Insert an element: 5
Enter choice: 2
Deleted element is: 6
Enter choice: 3
Queue is:
8    5
Enter choice: 4
Exited
```

11/1/24

WEEK -3

2. Circular Queue Implementation

```
#include <stdio.h>
int q[50], front = -1, rear = -1, size;
void display();
void enqueue();
void dequeue();
void main()
{
    int ch;
    printf("Enter no of elements :");
    scanf("%d", &size);
    while(ch != 4)
    {
        printf("1. Insert  2. Delete  3. Display  4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1:
                enqueue();
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
        }
    }
    printf("Exited");
}
```

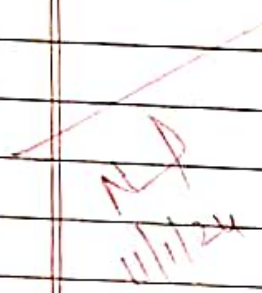
```
void enqueue()
{
    int item;
    if ((front == rear+1) || (front == 0 && rear == size-1))
        printf("Queue is full\n");
    else
    {
        if (front == -1)
            front = 0;
        printf("Enter the element : ");
        scanf("%d", &item);
        rear = (rear+1) % size;
        q[rear] = item;
    }
}

void dequeue()
{
    int ele;
    if (front == -1)
        printf("Queue is empty \n");
    else
    {   ele = q[front];
        if (front == rear)
        {   front = -1;
            rear = -1;
        }
        else
        {   front = (front+1) % size;
        printf("Deleted element = %d \n", ele);
}   }
```

```
void display()
{
    int i;
    if(front == -1)
        printf("Queue is empty");
    else
    {   printf("Front = %d \t", front);
        printf("Rear = %d \t", rear);
        printf("Queue is:");
        for(int i = front ; i != rear; i = (i+1)%size)
            printf("%d", q[i]);
        printf("%d\n", q[i]);
    }
}
```

OUTPUT :

```
Enter no. of elements : 5
1. Insert  2. Delete  3. Display  4. Exit
Enter your choice : 1
Enter element : 4
Enter your choice : 1
Enter element : 2
Enter your choice : 1
Enter element : 3
Enter your choice : 1
Enter element : 5
Enter your choice : 1
Enter element : 6
Enter your choice : 1
Queue is full
Enter your choice : 2
```

Deleted element = 4
Enter your choice : 2
Deleted element = 2
Enter your choice : 1
Enter element = 8
Enter your choice : 3
Front = 2 Rear = 0
Queue : 3568
Enter your choice : 4
Exited.



LAB - 4

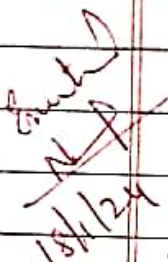
=) Insertion in a Linked List

```
#include <stdio.h>
#include <stdlib.h>
void push();
void append();
void display();
void insert_at_pos();
struct node
{
    int data;
    struct node *next;
};
struct node *head = NULL;

void main()
{
    int ch;
    while(ch!=6)
    { printf("1. Insert at begin 2. Insert at end 3. Insert in middle and 4. Exit \n");
      printf("Enter your choice :");
      scanf("%d", &ch);
      switch(ch)
      {
          case 1:
              push();
              break;
          case 2:
              append();
              break;
```

```
            case 3:
                insert_at_pos();
                break;
            case 4:
                display();
                break;
        }
    } printf
    printf(" Exited ");
}

void push()
{
    int data;
    struct node *new_node = (struct node *)
            malloc(sizeof(struct node));
    printf("Enter data: ");
    scanf("%d", &new_node -> data);
    new_node -> next = head;
    head = new_node;
}

void append()
{
    int data;
    struct node *last = head;
    struct node * new_node = (struct node*)
            malloc(sizeof(struct node));
    printf("Enter data: ");
    scanf("%d", &new_node -> data);
    new_node ->next = NULL;
```

```
        if(head == NULL)
            new_node = head;
        else
        { while (last ->next != NULL)
            last = last -> next;
          last = new_node;
        }
}

void insert_at_pos()
{
    int data;
    int pos;
    struct node *temp = head;
    struct node* ptr = (struct node*) malloc
                        (sizeof(struct node));
    printf("Enter data:");
    scanf("%d", &ptr->data);
    printf("Enter the position:");
    scanf("%d", &pos);
    if(pos==1)
    {
        ptr -> next = temp;
        ptr = head;
    }
    else
    {
        for(int i=0; i<pos-1; i++)
        {
        }   temp = temp -> next;
            ptr -> next = temp -> next;
            temp->next = ptr;
```

```
        ptr->next = NULL;
    }
  }
}

void display()
{
    struct node *p = head;
    printf("\n List : \n");
    while (p->next != NULL)
    {
        printf("%d\t", p->data);
        p = p->next;
    }
}
```



OUTPUT:

1. Insert from beginning 2. Insert at end
3. Insert at particular position 4. Display 5. Exit
Enter choice : 1
Enter the data to be inserted : 5
Enter choice : 2
Enter the data to be inserted : 4
Enter choice : 2
Enter data : 6
Enter choice : 3
Enter data : 7
Enter position : 2
List :
5 → 7 → 4 → 6 → NULL
Enter choice : 5
Exited

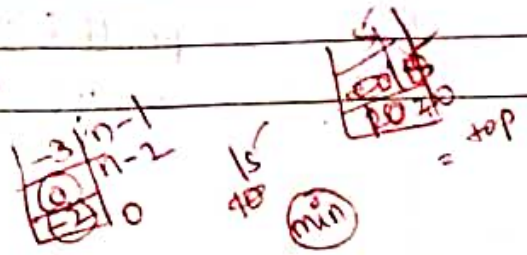
Bafna Gold
18/1/2024
Date: Page:

18/1

WEEK - 4

⇒ Deletion from a linked list.

```
#include <stdio.h>
#include <stdlib.h>
void pop();
void end_delete();
void delete_at_pos();
void display();
void append();
struct node
{
    int data;
    struct node *next;
};
struct node *head = NULL;

void main()
{
    printf("Insert the elements in list \n");
    append();
    printf("1. Delete from beginning \n 2. Delete
        at end \n 3. Delete at particular position
        \n 4. Display \n 5. Exit \n");
    int ch;
    while (ch!=5)
    {
        printf("Enter choice :");
        scanf("%d", &ch);
        switch(ch)
        {
```



```
        case 1:
            pop()
            break;
        case 2:
            end_delete();
            break;
        case 3:
            delete_at_pos();
            break;
        case 4:
            display();
            break;
        default:
            printf("Invalid choice");
            break;
    }
}
}

Void append()
{
    int data, n;
    printf("Enter no. of nodes:");
    scanf("%d", &n);
    for(int i=0; i<n; i++)
    {
        struct node *last = head;
        struct node *new_node;
        new_node = (struct node*) malloc
                (sizeof(struct node));
        printf("Enter the data:");
        scanf("%d", &new_node -> data);
```

```
        new_node -> next = NULL;
        if (head == NULL)
            head = new_node;
        else
        {   while (last -> next != NULL)
                last = last -> next;
            last -> next = new_node;
        }
    }
}

void pop()
{
    struct node * ptr;
    if (head == NULL)
        printf("List is empty\n");
    else
    {
        ptr = head;
        head = ptr -> next;
        free(ptr);
        printf("Node deleted from beginning\n");
    }
}

void end_delete()
{
    struct node * ptr;
    struct node * ptr1;
    if (head == NULL)
        printf("List is empty\n");
```

```
        else if (head -> next == NULL)
        {       free(head);
                head = NULL;
        }
        else
        {
                ptr = head;
                ptr1 = head;
                while ( ptr -> next != NULL)
                {       ptr1 = ptr;
                        ptr = ptr -> next;
                }
                ptr1 -> next = NULL;
                free(ptr);
                printf(" Node deleted from end \n");
        }
}

void delete_at_pos( )
{
    struct node *ptr;
    struct node *ptr1;
    int pos;
    printf(" Enter the position of deletion : \n");
    scanf(" %d ", &pos);
    ptr = head;
    for (int i = 0 ; i < pos - 1 ; i++)
    { if (ptr == NULL)
      { printf(" There are less elements \n");
        return ; }
      ptr1 = ptr;
      ptr = ptr -> next;
    }
```

```
        ptr1 -> next = ptr -> next;
        free(ptr);
        printf(" Node deleted from position \n");
    }

void display()
{
    struct node *p = head;
    printf(" List : \n");
    while ( p != NULL)
    {
        printf(" %d -> ", p -> data);
        p = p -> next;
    }
    printf("NULL \n");
}
```

18/1/24

OUTPUT :-

```
Enter no. of nodes : 5
Enter the ~~choice~~ data : 1
Enter the data : 2
Enter the data : 3
Enter the data : 4
Enter the data : 5
1. Delete from begining
2. Delete at end
3. Delete at particular position
4. Display
5. Exit
Enter choice : 1
Node deleted from beginning
```

Enter choice : 2
Node deleted from end
Enter choice : 4
List :
2 -> 3 -> 4 -> NULL
Enter choice : 3
Enter the position of deletion :
3
Node deleted from position 3
Enter choice : 4
List :
2 -> 3 -> NULL
Enter choice : 5
Exited

LEETCODE 1 - 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 = 5000000;
    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);
    }
    return st;
}
```

```
void minstackPush(MinStack * obj, int val)
{
    if(obj->top == obj->size-1)
        printf("Stack 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 value;
    if(obj->top == -1)
        printf("Underflow");
    else
    {
        value = obj->s[obj->top];
        obj->top--;
        printf("%d is popped \n", value);
    }
}
```

```
int minStackTop (MinStack* obj)
{
    int value = -1;
    if (obj -> top == -1)
        printf ("Underflow");
    else
    {   value = obj -> s[obj -> top];
        return value;
    }
}

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

25/1/24

WEEK - 5

1. WAP to implement singly linked list with operations : Sort the linked list, reverse the list and concatenation of two linked lists.

```
#include <stdio.h>
#include <stdlib.h>
void reverse();
void sort();
void concatenate();
void insert1();
void insert2();
void display();
void struct node
{
    int data;
    struct node *next;
};
struct node *head1 = NULL, *head2 = NULL;
void main()
{
    printf("Insert the elements in first list:\n");
    insert1();
    printf("Sorted list:\n");
    sort();
    display();
    printf("Reversed list:\n");
    reverse();
    display();
```

```
printf("Insert the elements of second
        list to concatenate :\n");
insert2();
concatenate();
printf("Concatenated list :\n");
display();
}

void sort()
{
    struct node *curr = head;
    struct node *ptr = NULL;
    int temp;
    while(curr != NULL)
    {
        ptr = curr -> next;
        while(ptr != NULL)
        {
            if(curr -> data > ptr -> data)
            {
                temp = curr -> data;
                curr -> data = ptr -> data;
                ptr -> data = temp;
            }
            ptr = ptr -> next;
        }
        curr = curr -> next;
    } free(ptr);
}
```

Emil
25/11/24

```
void reverse ()
{
    struct node *prev = NULL;
    struct node *ptr = NULL;
    while (head1 != NULL)
    {
        ptr = head1 -> next;
        head1 -> next = prev;
        prev = head1;
        head1 = ptr;
    }
    head1 = prev;
}

void concatenate ()
{
    struct node *temp = head1;

    if (head1 == NULL)
    {
        struct node *p = head2;
        while (p != NULL)
        {
            printf("%d -> ", p -> data);
            p = p -> next;
        }
        printf("NULL \n");
    }
    else
    {
        while (temp1 -> next1 != NULL)
            temp = temp -> next;
```

```
            temp -> next = head2;
        }
    }

void display ()
{
    struct node *p = head1;
    while (p != NULL)
    {
        printf("%d -> ", p -> data);
        p = p -> next;
    }
    printf("NULL \n");
}

void insert1()
{
    int data, n;
    struct node *last = head1;
    struct node * new_node = (struct node *)
            malloc(sizeof(struct node));
    printf("Enter ~~data~~ no. of nodes: ");
    scanf("%d", &n);
    for (int i=0; i<n; i++)
    {
        printf("Enter the data: ");
        scanf("%d", &new_node -> data);
        new_node -> next = NULL;
        while (last -> next != NULL)
            last = last -> next;
        last -> next = new_node;
    }
}
```

```
void insert2()
{
    int data, n;
    printf(" Enter no. of nodes:");
    scanf("%d", &n);
    for (int i=0; i<n; i++)
    {
        struct node *last2=head2;
        struct node * new_node2;
        printf(" Enter data:");
        scanf("%d", &new_node->data);
        while(last->next!=NULL)
            last=last->next;
        last->next=new_node;
    }
}
```

OUTPUT:

```
Insert the elements in first list:
Enter no. of nodes : 5
Enter the data : 3
Enter the data : 2
Enter the data : 5
Enter the data : 1
Enter the data : 4
Sorted list :
1 -> 2 -> 3 -> 4 -> 5 -> NULL
Reversed list :
5 -> 4 -> 3 -> 2 -> 1 -> NULL
Insert elements in second list :
Enter no. of nodes : 2
Enter the data : 6
Enter the data : 7
Concatenated list :
5 -> 4 -> 3 -> 2 -> 1 -> 6 -> 7 -> NULL
```

25/1/24

Bafna Gold

2. Stack implementation using linked list

```
#include <stdio.h>
void push();
void pop();
void display();
struct node
{
    int data;
    struct node * next;
};
struct node *head = NULL;

void main()
{   int ch
    printf("Enter : 1. Push 2. Pop 3. Display:");
    scanf("%d",&ch);
    while (ch != 4)
    {
        Switch (ch)
        {
            case 1 : push();
                     break;
            case 2 : pop();
                     break;
            case 3 : display();
                     break;
        }
    }
    printf("Exited");
}
```

Executed 25/1/24.

```
void push()
{
    int data, n;
    printf("Enter no of nodes:");
    scanf("%d", &n);
    for(int i=0; i<n; i++)
    {
        struct node *last = head;
        struct node * new_node
        = (struct node*) malloc
        (sizeof(struct node));
        printf("Enter the data:");
        scanf("%d", &new_node->data);
        new_node->next = NULL;
        while(last->next != NULL)
            last = last->next;
        last->next = new_node;
    }
}

Void pop()
{
    struct node *ptr;
    struct node * ptr1;
    if(head->next == NULL)
    {   free(head);
        head = NULL;
        printf("Element popped");
    }
    else
    {
        ptr = head;
```

```
        ptr1 = head;
        while (ptr -> next != NULL)
        {
            ptr1 = ptr;
            ptr = ptr -> next;
        }
        ptr1 -> next = NULL;
        free (ptr);
        printf (" Element popped successfully");
    }
}

void display ()
{
    struct node *p = head;
    printf ("Stack : \n");
    while (p != NULL)
    {
        printf ("%d -> ", p -> data);
        p = p -> next;
    }
    printf ("NULL \n");
}
```

OUTPUT :

1. Push
2. Pop
3. Display
4. Exit
Enter choice : 1
Enter the data : 5
Element pushed successfully
Enter choice : 1
Enter the data : 4
Element pushed successfully
Enter choice : 1
Enter the data : 3
Element pushed successfully
Enter choice : 3
List :
5 → 4 → 3 → NULL
Enter choice : 2
Element popped successfully
Enter choice : 3
List
5 → 4 → NULL
Enter choice : 4
Exited

25/1/24

Bafna Gold
Date: Page:

3. Queue Implementation using linked list

```
#include <stdio.h>
void enqueue();
void dequeue();
struct node
{
    int data;
    struct node *next;
};
struct node *head = NULL;
void main()
{
    printf("1. Enqueue 2. Dequeue 3. Display 4. Exit");
    int ch;
    while(ch != 4)
    {
        printf("Enter choice: ");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1: enqueue();
                    break;
            case 2: dequeue();
                    break;
            case 3:
                    display()
                    break;
        }
    }
}
```

```
void enqueue()
{
    int data;
    struct node *last = head;
    struct node *new_node;
    new_node = (struct node*) malloc
        (sizeof(struct node));
    printf("Enter the data:");
    scanf("%d", &data);
    new_node -> data = data;
    new_node -> next = NULL;
    if(head == NULL)
        head = new_node;
    else
    {   while(last -> next != NULL)
            last = last -> next;
        last -> next = new_node;
    } printf("Node added successfully\n");
}

void dequeue()
{
    struct node *ptr;
    if (head == NULL)
        printf("List is empty\n");
    else
    {   ptr = head;
        head = ptr -> next;
        free(ptr);
        printf("Node deleted from beginning\n");
    }
}
```

```
void display()
{
    struct node *p = head;
    printf("\n");
    while(p != NULL)
    {
        printf("%d -> ", p->data);
        p = p->next;
    }
    printf("NULL\n");
}
```

OUTPUT:

```
Enter:
1.Enqueue
2. Dequeue
3. Display
Enter choice: 1
Enter data: 1
Enter choice: 1
Enter data: 2
Enter choice: 1
Enter data: 3
Enter choice: 3
Queue:
    1 -> 2 -> 3 -> NULL
Enter choice: 2
Node deleted from beginning
Enter choice: 3
Queue:
2 -> 3 -> NULL
Enter choice: 4
Exited
```

25/1/24

LEETCODE 2 - Reversed Linked List

```
struct ListNode* reverseBetween (struct
ListNode* head , int left , int right)
{
    if (head == NULL || left == right)
        return head;

    struct ListNode* dummy = (struct ListNode*)
        malloc (sizeof (struct ListNode));
    dummy -> next = head;

    struct ListNode* preleft = dummy;
    for(int i=1 ; i<left ; i++)
    {       preleft= preleft ->next;
    }

    Struct ListNode * curr = preleft -> next;
    struct ListNode* prev = Null;
    struct ListNode* next = NULL;

    for (int i=0 ; i<=right - left ; i++)
    {   next = curr -> next;
        curr -> next = prev;
        prev = current;
        current = next;
    }
    preleft -> next -> next = current;
    preleft -> next = prev;
    if(left == 1)
    {   head = dummy -> next;
    }
```

```
        free(dummy);
        return head;
    }
```

INPUT

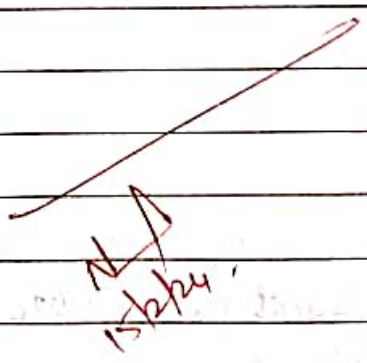
[1, 2, 3, 4, 5]

left = 2

right = 4

OUTPUT

[1, 4, 3, 2, 5]



15/2/24.

1/2/24

LAB - 7

WAP to implement doubly linked list with primitive operations

(a) Create a doubly linked list.

(b) Insert a new node to the left of the node

(c) Delete the node based on specific value.

```
#include <stdio.h>
struct node
{
    int data;
    struct node *prev;
    struct node *next;
};
struct node *head = NULL;
Void insert();
void delete();
void display();
void main()
{
    int ch;
    printf("Enter 1. Insert \n 2. Delete \n 3. Display \n
                4. Exit \n");
    while (ch != 4)
    {
        printf("Enter choice:");
        scanf("%d", &ch);
        switch (ch)
        {
            case 1 : insert();
                     break;
```

```
        case 2: delete ();
                break;
        case 3: display ();
                break;
        }
    }
    printf("Exited");
}

Void insert()
{   printf("Enter position: "); scanf("%d", &pos);
    struct node *new = (struct node*) malloc
                (sizeof(struct node));
    printf("Enter the data:");
    scanf("%d", &new->data);
    if(head == NULL)
    {
        new -> prev = NULL;
        new -> next = NULL;
        head = new
        printf("Node inserted \n");
    }
    else { for(i=0; i<pos-1; i++)
    { ptr = ptr -> next; }
    {   new -> prev = ptr -> prev;
        new -> next = ptr;
        ptr -> prev -> next = new;
        ptr -> prev = new;
        printf("Node inserted \n");
    }
}
```

```
void delete()
{
    int val;
    printf("Enter the value: ");
    scanf("%d", &val);
    struct node *ptr = head;
    if(head -> data == val)
    {
        head = ptr -> next;
        free(ptr);
        printf("Node deleted \n");
        return;
    }
    while (ptr -> data != val)
    {
        ptr = ptr -> next;
        if (ptr -> next == NULL)
        {
            ptr -> prev -> next = NULL;
            free(ptr);
            printf("Node deleted \n");
            return;
        }
    }
    ptr -> prev -> next = ptr -> next;
    ptr -> next -> prev = ptr -> prev;
    free(ptr);
    printf("Node deleted \n");
}
```

```
void display()
{
    struct node *p=head;
    while(p!=NULL)
    {
        printf("%d -> ",p->data);
        p=p->next;
    }
    printf("NULL\n");
}
```

OUTPUT :

```
Press 1. Insert
      2. Delete
      3. Display
Enter choice: 1
Enter data : 4
Enter choice: 1
Enter data : 5
Enter position: 1
Enter choice : 1
Enter position: 2
Enter data : 3
Enter choice : 3
5 -> 3 -> 4 -> NULL
Enter choice : 2
Node deleted
Enter choice : 3
5 -> 4 -> NULL
Enter choice: 4
Exited
```

15/2/24

LAB - 8

WAP (i) To construct a binary search tree.
(ii) To traverse tree using all the methods, i.e. in-order, preorder and postorder
(iii) To display the elements in the tree

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

struct node
{
    int data;
    struct node *left;
    struct node *right;
};
struct node *root=NULL;
void create()
{
    int val;
    struct node *new_node = (struct node*)
                malloc(sizeof(struct node));
    Struct node *ptr = root;
    struct node *ptr1 = NULL;
    printf("Enter data:");
    scanf("%d", &val);
    new_node -> left = NULL;
    new_node -> right = NULL;
    new_node -> data = val;
    if(root == NULL)
    {   root = new_node;
    }
```

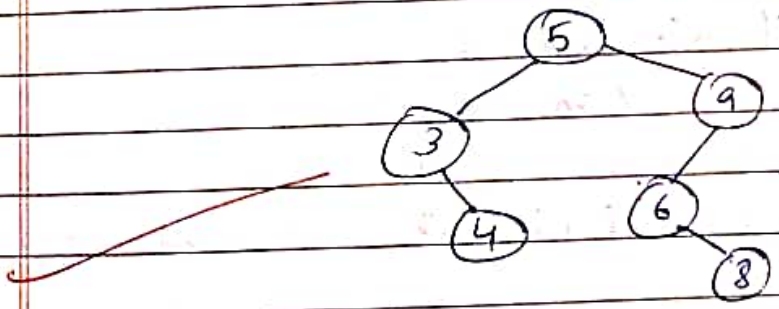
```
else
{
    while (ptr != NULL)
    {   ptr1 = ptr;
        if (ptr -> data > val)
            ptr = ptr -> left;
        else
            ptr = ptr -> right;
    }
    if (ptr1 -> data > val)
        ptr1 -> left = new_node;
    else
        ptr1 -> right = new_node;
}
}

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

void pre_order (struct node *ptr)
{
    if (ptr != NULL)
    {   printf ("%d ", ptr -> data);
        pre_order (ptr -> left);
        pre_order (ptr -> right);
}   }
```

15/2/24

```
void post-order (struct node *ptr)
{
    if (ptr != NULL)
    {
        post-order (ptr -> left);
        post-order (ptr -> right);
        printf ("%d ", ptr -> data);
    }
}

void main()
{
    int n;
    printf ("Enter no. of nodes: ");
    scanf ("%d ", &n);
    for (int i=0 ; i<n ; i++)
    {
        create();
    }
    printf ("\n Inorder: \n");
    inorder (root);
    printf ("\n PreOrder: \n");
    pre-order (root);
    printf ("\n PostOrder: \n");
    post-order (root);
}
```

OUTPUT

```
Enter no. of nodes : 6
Enter data : 5
Enter data : 3
Enter data : 4
Enter data : 9
Enter data : 6
Enter data : 8
Inorder :
3 4 5 6 8 9
Preorder :
5 3 4 9 6 8
Postorder :
4 3 8 6 9 5
```



15/2/24

LeetCode - 4

Rotate List

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

I/p -> head = [1, 2, 3, 4, 5] , k = 2

o/p -> [4, 5, 1, 2, 3]

15/2/24

Bafna Gold
Date: Page:

Leet Code - 3

```
struct ListNode * splitListToParts(struct ListNode*
    head, int k, int * returnSize){
    int length = 0;
    struct listnode *ptr = head;
    int partSize = length/k;
    int extraNodes = length % k;

    while (ptr != NULL){
        length++;
        ptr = ptr->next;
    }

    struct ListNode **result = (struct ListNode**)
        malloc(k * sizeof(struct ListNode*));

    for(int i=0; i<k; i++)
    {
        result[i] = head;
        int currentPart = partSize +
            (extraNodes-- > 0 ? 1 : 0);
        for(int j=0; j<currentPart-1; j++){
            if(head != NULL){
                head = head->next;
            }
        }

    if(head != NULL){
        struct ListNode *temp = head;
        head = head->next;
        temp->next = NULL;
    }
```

```
    }
    *returnSize = k;
    return result;
}
```

Input -

head = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

k = 3

output -

[[1,2,3,4], [5,6,7], [8,9,10]]

22/2/24

LAB - 9

BFS & DFS traversals.

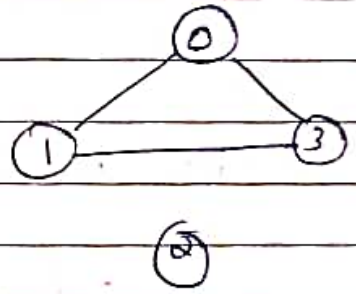
```
#include <stdio.h>
int queue[100];
int front = 0 , rear = 0;
int a[10][10];
void enqueue (int var)
{
    queue[rear] = var;
    rear++;
}
void dequeue ()
{   front++;
}

void bfs (int n)
{
    int visited[10] = {0};
    equeue(0);
    visited[0] = 1;
    while (front != rear)
    {
        int current = queue[front];
        printf("%d" , current);
        dequeue ();
        for(int i=0 ; i<n;i++)
        {
            if ( a[current][i] && !visited[i])
            {
```

```
            visited[i] = 1;
            enqueue(i);
        }
    }
}

void dfs(int a[][10], int n, int start,
         int visited2[10]) {
    visited2[start] = 1;
    printf("%d ", start);
    for(int i=0; i<n; i++) {
        if(a[start][i] && !visited2[i]){
            dfs(a, n, i, visited2);
        }
    }
}

void main()
{
    int n;
    int start = 0;
    int visited2[10];
    printf("Enter no. of vertices: ");
    scanf("%d", &n);
    printf("Enter adjacency matrix:\n");
    for(int i=0; i<n; i++)
    {
        for(int j=0; j<n; j++)
            scanf("%d", &a[i][j]);
    }
```

```
printf("BFS traversal: \n");
bfs(n);

for(int i=0; i<10; i++){
    visited2[i] = 0;
}
printf("DFS traversal: \n");
dfs(a, n, start, visited2);
}
```

OUTPUT -

```
Enter no of vertices: 4
Enter adjacency matrix:
0 0 1 0 1
0 1 0 0 1
0 0 0 0 0
1 1 0 0
BFS traversal:
1 0 3
DFS traversal:
0 1 3
```



Disconnected graph

Ex. 1

22/2/24

29/2/24

Bafna Gold
Date: Page:

HackerRank

```
typedef struct TreeNode {
    int data;
    struct TreeNode * left;
    struct TecNode * 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 [indexes-rows + 1];
  nodes [1] = root;

  for(int i = 0; i < indexes-rows ; 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 -> left = NULL;
          curr -> left -> right = NULL;
          nodes [indexes [i][0]] = curr -> left;
      }
  }
  return root;
}
```

```
int ** swapNodes (int indexes_rows, int
indexes_columns, int ** indexes, int queries_
count, int * queries, int * result_rows,
int * result_columns)
{
    int ** result = (int **) malloc (queries_
            count * sizeof (int *));
    *result_rows = queries_count;
    *result_columns = indexes_rows;

    TreeNode * root = buildTree (indexes_rows,
        indexes_columns, indexes);

    for (int i=0 ; i< queries_count ; i++)
    {
        int k = queries[i];
        swapSubtrees (root, k, k);

        int * traversal = (int *) malloc
            (indexes_rows * sizeof (int));
        int index=0;
        inOrderTraversal (root, traversal, &index);
        result[i] = traversal;
    }

    return result;
}
```

~~OUTPUT~~ :-

INPUT -

3

2 3

-1 -1

-1 -1

2

1

1

OUTPUT

3 1 2

2 1 3

29/12/24

29/2/24

LAB - 10

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_MEMORY_LOCATIONS 10
int hashTable[MAX_MEMORY_LOCATIONS] = {0};
int status[MAX_MEMORY_LOCATIONS] = {0};

void initializeHashTable(int size){
    for(int i=0; i<size; i++){
        status[i] = 0;
    }
}

int hashFunction(int key, int size){
    return key % size;
}

int linearProbe(int hashValue, int attempt,
                int size){
    return (hashValue + attempt) % size;
}

void insertEmployee(int size, int key){
    int hashValue = hashFunction(key, size);
    int index = hashValue;
    int attempt = 1;
    while(status[index] == 1){
        index = linearProbe(hashValue, attempt,
                size);
        attempt++;
    }
    hashTable[index] = key;
```

```
    status[index] = 1;
}

void displayHashTable(int size){
    printf("\n Hash Table:\n");
    printf(" Index\t key\n");

    for(int i=0; i<size; i++){
        printf("%d\t", i);
        if(status[i]==1){
            printf("%d\n", hashTable[i]);
        }else{
            printf(" Empty\n");
        }
    }
}

int main(){
    int m = MAX_MEMORY_LOCATIONS;
    initializeHashTable(m);
    int n;
    printf("Enter the number of employee
            records: ");
    scanf("%d", &n);
    int keys[n];

    printf("Enter the employee keys:\n");
    for(int i=0; i<n; i++){
        printf(" Employee %d: ", i+1);
        scanf("%d", &keys[i]);
    }
    for(int i=0; i<n; i++){
```

```
        insertEmployee(m, keys[i]);
    }

    displayHashTable(m);

    return 0;
}
```

OUTPUT -

```
Enter the number of employee records: 6
Enter the employee keys:
Employee 1 : 85
Employee 2 : 56
Employee 3 : 23
Employee 4 : 35
Employee 5 : 43
Employee 6 : 11

Hash Table:
Index    Key
0        Empty
1        11
2        Empty
3        23
4        43
5        85
6        56
7        35
8        Empty
9        Empty
```