INDEX

Name: SUFIYAN.. DESAI Std: ____ Sec: D

Roll No. ____ Subject: ____ School/College: DS observation

School/College Tel. No. ____ Parents Tel. No. ____

Sl. No.	Date	Title	Page No.	Teacher Sign / Remarks
lab 1	7/12/23	Stack implementation		
lab 2	28/12/23	1. Infix to Postfix 2. Postfix evaluation		
lab 3	11/1/24	1. circular queue 2. linear queue		
lab 4 lab 5	18/1/24	1. Singly LL insertion 2. Singly LL deletion 3. Leetcode (Minstack)	10	
lab 6	25/1/24	1. operations on LL 2. LL as stack 3. LL as queue		
LAB 7	1/2/24	1. Doubly LL 2. Leet code		
LAB 8	16/2/24	1. Binary Search tree 2. Leetcode		
LAB-9	22/2/24	1. BFS 2. DFS 3. Leetcode	10	
LAB-10	29/2/24	1. Hashing		

21/12/23

own

Date __/__/__
Page _____

LAB - 1

→ Write a program for stack implementation :

```
#include <stdio.h>
# define MAX 4

int top ==-1;
int stack_arr [MAX] ;

void push (int data)
{
    if ( top == MAX-1 )
    {
        pf (" stack overflow");
    }

    else {
            pf (" enter element to be added");
            scanf ("%d", &data);

            top = top+1 ;
            stack_arr [top] = data;
    }
}

void pop ()
{
        int value;
    if (top ==-1)
    {
        pf ("underflow");
    }
```

```
        value = stack_arr [top] ;
            top = top -1 ;
            return ;
    }
}

void display ()
{
    if (top == -1)
    {
        pf ("underflow")
    }
else{
    pf ("Elements in stack are")
    for (int i = top ; i >= 0 ; i ++)
    {
        pf ("%d" stack_arr[i]);
    }
}

int main ()
{
    int choice;
    while (1)
    {
        pf ("1. Push \n 2. Pop \n 3. display \n
                4. Exit");
        pf ("Enter your choice");
        scanf ("%d", &choice);
```

```
switch (choice)
{
    case 1:  push ( ) ;
             break ;

    case 2:  pop ( ) ;
             break ;

    case 3:  display ( ) ;
             break ;

    case 4:  exit (0) ;
  }
}
}
```

<u>Output</u> : (SUFIYAN 1BM22EC261)

1. Push the element
2. Pop element
3. Display
4. Exit

Entry your choice : 1
Enter the element to be pushed : 4

1. Push the element
2. Pop element
3. Display
4. Exit

Entry choice : 2
Entr element to pop : 4

1. Push element
2 Pop element
3 Display
4 Exit
Enter your choice : 4

Date ___/___/___
Page. _____

LAB-2

→ WAP to convert infix to Postfix evaluation:

```
#include <stdio.h>
#include <stdlib.h>
# define MAX 100

char stack-arr [MAX];
int top = -1;

void push (char x)
{
    if (top == MAX-1)
    {
      pf ("overflow");
    }

    else { top = top +1;
           stack-arr [top] = x;
    }
}

void pop ()
{
    char c;
    if (top == -1)
    {
      pf ("underflow");
    }
    else {
           x = stack_arr[top];
```

```
        return ;
    }
}

int precedence (char symbol)
{
    if ( symbol == '^')
        return 3;
    else if ( symbol == '*' || symbol == '/')
        return 2;

    else if ( symbol == '+' || symbol == '-')
        return 1;
    else
        return -1;
}

void infix_to_postfix (char infix [],
                       char postfix [])
{
    int i=0, j=0;
    char symbol , temp;
    push ('#') ;

    while ( symbol = infix [i++] != '\0')
    {

        if (symbol == '(')
            push (symbol) ;
        else if ( isalnum (symbol))
            postfix [j++] = symbol ;
```

Date __/__/__
Page _____

```
else if (symbol == ')')
{
  while stack_arr [top] != '('
  {
    postfix [j++] = pop();
  }

  temp = pop();
}

else
{ while (precedence (stack_arr [top]) >=
          precedence (symbol))
  {
    postfix [j++] = pop();
  }
  push (symbol);
}
}

while (stack [top] != '#')
{
  postfix [j++] = pop();
}

postfix [j] = '\0';
}


int main()
{
  char infix [MAX], postfix [MAX];
  pf ("Enter infix exp ");
```

```
scanf ("%s", infix);
infix-to-postfix (infix, postfix);
pf ("The postfix is %s", postfix);
    return 0;
}
```

Output : SUFIYAN 1BM22EC261

Enter a infix expression :
a*b+c*d-e

The postfix exp is : ab* cd *+e-

→ WAP to evaluate postfix expressions:

```
# include <stdio.h>
# define max 20

int stack [Max];
int top = -1;

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

int pop ()
{
    return stack [top --] ;
}

void main ()
{
    char postfix (MAX];
  pf ("enter postfix exp");
  scanf ("%d, postfix);

    int res = 0 , a, b;
  for ( int i = 0 ;  i< strlen (postfix); i++)
{

    if ( isalnum ( postfix [i]))
        push (postfix[i] - '0') ;
  else
    {       b = pop ();
            a = pop ();
```

```
switch (postfix[i])
{
    case '+': push (a+b);
              break;

    case '-': push (a-b);
              break;

    case '*': push (a*b);
              break;

    case '/': push (a/b);
              break;

    case '^': push (a^b);
              break;
    }
  }
}

res = pop();
pf ("%s = %d", postfix, res);
}
```

<u>Output:</u> SUFIYAN 1BM22EC261

Enter postfix exp: 23 * 31 * + 5 -

23 * 31 * + 5 - = 4

Date __/__/__
Page _____

LAB - 3 (Linear Queue)

```
#include <stdio.h>
# define size 30

int queue [size] ;
  int front = -1;
  int rear = -1;

void insert (int a )
{
    if (rear == size-1)
    {
      printf (" overflow") ;
      return ;
    }
    else
    {
      if (front == -1)
          front = 0 ;
      queue [++rear] = a ;
    }
  }

void delete ()
{
    if (front == -1 || front > rear)
    {
        printf (" Queue empty ") ;
    }
    else
    { front ++ ;
    }
  }
```

```
void display () {
    if ( front == -1)
    {
        printf ("queue empty");
        return;
    }
    printf ("Queue");
    for (int i = front ; i <= rear;
            i++)
    {
        printf (" %d", queue [i]);
    }
}

void main ()
{
    int choice;
    int a;
    while (1)
    {
        printf ("1. Insert \n 2. delete \n
                3. display \n");
        scanf ("%d", &choice);
        Switch (choice)
        {
        case 1 : printf ("Enter a element");
                 scanf ("%d", &a);
                 insert (a)
                 display ();
```

Date __/__/__
Page. _____

```
        break;

    case 2:     delete();
                display();
                break;

    case 3:     display();
                break();
    }
}
}
```

Output :

```
1.  Insert
2.  delete
3   display
    choice : 1
    Enter an element : 7
    Queue : 7.

1   insert
2   delete
3   display
    choice : 1
    Enter an element : 8
    Queue : 7 8

1.  Insert
2.  delete          Enter choice : 2
3.  display
    Queue 8
```

Date __/__/__
Page

LAB - 4 (Circular Queue)

```
#include <stdio.h>
#define size 5

int queue [size];
int front = -1;
int rear = -1;

void enqueue (int a)
{
    if ( front == rear +1 ) || (front == 0
                                && rear == size-1))
    {
        pf ("Queue overflow");
        return;
    }
    else
    {
        if (front == -1)
            front = 0;
        rear = (rear+1) % size;
        queue [rear] = a;
    }
}

Void dequeue ()
{
    if (front == -1)
    {
        pf ("Empty");
    }
```

Date __/__/__
Page _____

```
else
{
    inta = queue [front];
    if (front == rear)
    {
        front = -1;
        rear = -1;
    }
    else
    {
        front = (front +1) % size;
    }

    printf (" Deleted element = %d, a);
    return (a);
}
}

void display ()
{
    if (front == -1)
    {
        printf ("Queue empty");
        return;
    }
    else
    {
```

```
int i;
pf (" Front = %d ", front);
pf (" Items = ");
for (i = front [i] != rear ;
        i = (i+1) % size)
{
    pf (" %d ", queue[i]) ;
}

pf (" %d ", queue [i]);
pf (" rear = %d ", rear);
}
}

void main ()
{
    int choice;
    int a;

while (1)
{
    pf ("1. Insert \n 2. delete \n 3. display
    ");
    scanf ("%d", &choice);
switch (choice)
{
    case 1: pf (" Enter element ");
    scanf ("%d", &a);
    enqueue (a);
    display ();
    break;
```

```
case 2:   dequeue ();
          display ();
          break;

case 3:   display ();
          break;
```

Output :

```
1. Insert
2. delete
3. display
Choice : 1
Enter element: 2

Front = 0
items = 2
rear = 0

1. Insert
2. delete
3 display

Choice:1
Enter element : 3
Front = 0
items = 2 3
rear = 1
```

OWN

Date / /
Page

LAB - 4

→ WAP to insert element at various positions in a linked list :

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

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

void push ( struct node ** head_ref, int
{                                new_data )
    struct node * newnode = (struct node*)
        malloc (size of (struct node));

        newnode -> data = newdata;
        newnode -> next = *(head_ref);
        (*head_ref) = newnodie;
    }

void append ( struct node ** head_ref, int new_data)
{
    struct node*) newnode = (struct node *) malloc
                    (size of (struct node));

        struct node * last = (* head_ref);
        newnode -> data = newdata ;
        new node -> next = NULL ;
        if ( *head_ref == NUU)
        {
            (*head_ref) = newnode ;
            return ;
```

Date __/__/__
Page _____

```
}
    while (last.next != NULL)
    {
        last = last -> next;
        last -> next = newnode;
        return;
    }

void specpos ( int data, int position)
{
struct node * new node = (struct node*) malloc
        (size of (struct node))

    newnode -> data = newdata;
        int i;
    struct node * temp = head;
    if (position == 1)
    {
        newnode -> next = temp;
            head = ptr;
            return;
    }
    for ( i=1 ; <position-1 ; i++)
    {
        temp = temp->next;
    }

        new node -> next = temp -> next;
            temp -> next = newnode;
    }
```

```
void display (structnode * node)
{
    while ( node != NULL)
    {
        pf ("%d", node -> data);
        node = node -> next;
    }
}

int main ()
{
    struct node * head = NULL;
    int choice, value, position;
    while (1)
    {
        pf ("1. Push \n 2. append \n 3. specpos \n
            4. display \n 5. exit.");
        pf ("Enter your choice");
        scanf ("%d", & choice);

        switch (choice)
        {
            case 1:
            pf ("Enter element to push");
            scanf ("%d", &value);
            push (&head, value);
            break;

            case 2:
            pf ("Enter the element to append");
            scanf ("%d", & value);
            append (&head, value);
            break;
```

Date __/__/__
Page _____

```
case 3:
    pf ("Enter element to be inserted at specpos");
    scanf ("%d, &value);
    pf ("Enter pos");
    scanf ("%d", &position");
    specpos (&head, value);
    break;

case 4: display (Head);
        break;

case 5 : exit (0);
}
pf ("The created list is ");
display(head);
}
}
```

Output :

```
1. Push
2. append
3. specpos
4. display
5. exit
Enter your choice : 1
Enter ele to push : 4
created list is : 4
1. Push
2. Append
3. specpos
4 display
```

5. exit
Enter your choice : 2
Enter ele to append : 8
Created list is : 4 8

Date __/__/__
Page _____

LAB5

→ WAP for deletion of elements :

```
#include <stdio.h>
struct node {
        int data;
        struct node * next;
        } ;
        struct node *head;
void pop()
{
    struct node *ptr;
    if ( head == NULL)
    {
        pf ("empty");
    }
    else {
        head = ptr;
        head = ptr.next;
        free (ptr);
        return;
    }
    }

void append ()
{
    struct node* ptr * ptr1;
    if (head == NULL)
    {
        pf ("list empty");
    }
    else if (head.next == NULL)
```

```
{       free (head);
        head = NULL;
    pf ("Node is deleted");
}

else {
        ptr = head ;
    while (ptr->next != NULL)
    {
        ptr1 = ptr;
        ptr = ptr.next;
    }

        ptr1.next = NULL;
        free(ptr);
    pf ("Last node deleted");
  }
}


void delspec ()
{
    struct node *ptr *ptr1;
        int loc, i;
    scanf ("%d", &loc);
        ptr = head;
    for (i=0 ; i<loc ; i++)
    {
        ptr1 = ptr;
        ptr = ptr.next;
    if ( ptr == NULL)
```

Date __/__/__
Page _____

```
    Pf ("%d", loc);
    return;
}
}
    ptr1.next = ptr.next;
        free(ptr);
    }
    Pf ("deleted node at %d", loc);
    }

void push()

void display ()
{
    struct node *ptr;
        ptr = head;
    if (ptr == NULL)
    {
        Pf ("Nothing to print");
    }
    else
    {
        Pf ("\n");
        while (ptr != NULL)
    {
        Pf ("%d", ptr.data);
            ptr = ptr.next;
    }
    }
}
```

```
int main ()
{
  int choice, value;
  while(1)
  {
    printf ("1. Push 2. Pop 3. append
            4 display 5. spec 6. exit");
    scanf ("%d", &choice);

    switch (choice)
    {
      case 1:
        pf ("Enter");
        scanf ("%d", &value);
        push (&head, value);
        pf ("created list is");
        display ();
        break;

      case 2:
        pop ();
        break ();

      case 3:
        append ();
        break;

      case 4:
        display ();

      case 5: specpos ();
              break;
      case 6: exit(0);
    }
```

Date __/__/__
Page. _____

Output :

1. Push
2. Pop
3. append
4. display
5. specpos
6. exit

Enter your choice : 1
The created list is : 1

1. Push
2. Pop
3. append
4. display
5. specpos
6. exit

Enter your choice : 2
List is : 1 2

1. Push
2. Pop
3. append
4. display
5. specpos
6. exit

Enter your choice : 4
List is : 1

18/1/24

Date __/__/__
Page ____

Leetcode (Minstack)

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

typedef struct
{
    int value;
    int min;
} struct node;

typedef struct

{   struct node * array;
    int capacity;
    int top;
}
    min stack;

MinStack * minStackCreate ()
{
    MinStack * Stack = (MinStack *)
        malloc (Size of (Minstack));

    Stack -> capacity = 10;
    Stack -> capacity = 10;
    Stack -> array = (Stack Node *)
        Malloc ( stack -> capacity *
        size of (struct node));

    Stack -> top = -1;
    return Stack;
}
```

Date __/__/__
Page _____

```
void minStackPush (MinStack * obj,
                   int val)
{
    if (obj -> top == obj -> capacity-1)
    {

        obj -> capacity* = 2;
        object -> array = (StackNode *)
        realloc (obj -> array, obj -> capacity
            * size of (StackNode));
    }


    Stack Node newNode;
    newnode.value = val;
    newnode.min = (obj -> top == -1) ?
        val : (val < obj -> array [obj -> top] min) ?
    val : obj -> array [obj -> top] min;

    obj -> array [++ (obj -> top)] = newNode;
}

void minStackPop (MinStack * obj)
{
    if (obj -> top != -1)
    {
        obj -> top--;
    }
}
```

Date __/__/__
Page

```
int minStackTop (MinStack *obj)
{
    if (obj->top != -1)
    {
        return obj->array[obj->top].value;
    }

    return -1;
}

int minStackGetMin (MinStack *obj)
{

    if (obj->top != -1)
    {
        return obj->array[obj->top].min;
    }
    return -1;
}


void minStackFree (minStack *obj)
{

    free (obj->array);
    free (obj);

}
```

LAB - 6 (operations on LL)

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

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

typedef Struct Node Node;

Node * create Node (int value)
{
    Node * newNode = (node *) malloc
                        (size of (Node));

    newnode -> data = value;
    newnode -> next = NULL;
    return newNode;
}

void displayList( Node *head)
{

    while (head != NULL)
    {
        pf ("%d -> ", head -> data);
        head = head -> next;
    }

    Pf (" Null ");
}
```

Date __/__/__
Page

```
Node * sortlist (Node *head)
{
    if (head == NULL || head->next = NULL)
    {
        return head;
    }

    int swapped ;
    Node * temp ;
    Node * end = NULL;

do
{
    swapped = 0;
    temp = head;
    while (temp -> next != end)
{
    if (temp -> data > temp -> next -> data)
    {
        int tempData = temp -> data;
        temp -> data = temp -> next -> data
        temp -> next -> data = tempData;
        swapped = 1;
    }

        temp = temp -> next;
    }

    end = temp;
}
    while (swapped)
        return head;
}
```

Date __/__/__
Page _____

```
Node * reverseList (Node * head)
{
    Node * prev = NULL;
    Node * current = head;
    Node * nextnode = NULL;

while ( current != NULL)
{
    nextNode = current -> next;
    current -> next = prev;
    prev = current;
    current = nextnode;
}

return prev;
}

Node * concatenation (Node * L1,
                      Node * L2)
{
    if (L1 == NULL)
        return L2;

Node * temp = L1;
while (temp -> next != NULL)
{
    temp = temp -> next;
}
    temp -> next = L2;
        return L1;
}
```

Date __/__/__
Page ____

```
int main()
{
    node *L1 = createNode (3);
    L1 -> next = create Node (1);
    L1 -> next -> next = create Node (4);

    Node * L2 = createNode (2);
    L2 -> next = createNode (5);

    printf ("original list\n");
    displaylist (L1);

    printf ("og List 2);
    display list (L2);

    L1 = SortList (L1);
    pf ("Sorted list 1:");
    display list (L1);

    L1 = sortList (L1);
    pf ("Sorted List\n");
    displayList (L1);

    L1 = reverseList (L1);
    pf ("reversed list");
    displayList (L1);

    Node * concatenated = Concatenated
        Lists (L1, L2);
    pf ("Concat list :");
    display list (concat list);

        return 0;
}
```

Date __/__/__
Page _____

Output :

1
Enter data : 30

1
Enter data: : 10

1
Enter data : 40
2

4
10 30 40

3 4
40 30 10

25/1/24

Date __/__/__
Page ____

Stack implementation of LL

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

struct node {
        int data;
        struct node * next;
    };
    struct node* head; *newnode, *temp,
        * temp1;

Void push ()
{
  newnode = (struct node*) malloc (sizeof
                                (struct node));
  pf (" Enter data to push ");
  scanf (" %d ", &newnode -> data);
    newnode -> next = head;
      head = newnode;
}

    void display ()
{
    pf (" The created stack is ");
        temp = head;
    while (temp != NULL)
  {
      pf (" %d ", temp -> data);
        temp = temp -> next;
    }
}
```

Date __/__/__
Page _____

```
void pop()
{
    if (head == NULL)
        pf ("stack empty) ;
else
    {
        head = head -> next ;
    }
}

int main ()
{
    int ch;
    while (1)
    {
        pf ("1. Push \n 2. Pop \n
                3. Display \n 4. Exit");
        scanf ("%d", &ch);
    switch (ch)
    {
    case 1 :    push() ;
                display ();
                break;

    case 2 :    pop();
                display ();
                break;

    case 3 :    display ();
                break ;
```

```
        case 4: exit (0);
        }
    }
}
```

Output:

```
1. Push
2. Pop
3. display
4. exit
choice : 1
Enter data : 20

1. Push
2. Pop
3. display
4. Exit
choice : 1
Enter data : 25

The stack is 20 25
```

25/1/24

Date __/__/__
Page _____

Queue Implementation
of LL

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

struct node {
        int data;
        struct node* next;
    };
    struct node *head, *newnode,
        *temp, *temp1;

void pushqueue ()
{
    newnode = (struct node*) malloc
            (sizeof (struct node));
    pf ("Enter data to push");
    scanf ("%d", &newnode -> data);
    newnode -> next = head;
        head = newnode;

}

void display ()

{   pf ("The queue is");
    temp = head;
    while (temp != NULL)
    {
        pf ("%d", temp -> data);
            temp = temp -> next;
    }
}
```

Date __/__/__
Page ____

```
void popqueue ()
{
        if (head == NULL)
            pf ("queue empty");

    else
    {
        temp = temp1 = head;
        while (temp -> next != NULL)
        {
            temp1 = temp;
            temp = temp -> next;
        }

        temp1 -> next = NULL;
        free (temp);
    }
}

int main ()
{
    int ch;
    while (1)
    {
        pf ("1. Push \n 2. Pop \n
            3. display ");
    scanf ("%d", &ch);

    switch (ch)
    {
```

Date __/__/__
Page _____

```
Case 1: pushqueue();
        display();
        break;

case 2: popqueue();
        display();
        break;

Case 3: display();
        break;
}
}
}
```

Output:

```
1. Push
2 Pop
3 display
Choice : 1
Enter element : 50

1 Push
2 Pop
3 display
Choice : 1
Element : 60

The created list is 50 60
```

1/2/24

Date __/__/__
Page

LAB - 7 - Doubly LL

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

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

struct node * createNode ( int data)
{
    struct node * newnode = (struct node*)
malloc (size of (StructNode));

    if (newNode == NULL)
    {
        pf ("Memory allocation fail");
            exit (1);
    }

    newnode -> data = data;
    newnode -> prev = NULL;
    newnode -> next = NULL;
    return newnode ;
}

void insert left ( struct node * head,
    struct Node * target, int data)
{
    struct Node* newnode = createNode (data);
    if ( target -> prev -> next = newnode ;
        newnode -> prev -> = target -> prev;
    }
```

```
else {  head = newNode;
    }
    newnode -> next = target;
    target -> prev =  newNode;
}
void deleteNode (struct Node* hed, int
                                value)
{
    struct Node* current = head;
    while (current != NULL)
    {
        if (current -> data == value)
        {
            if (current -> prev != NULL)
            {
                curret -> prev -> next = current
                                        -> next;
            }
else
{
    head = current -> next;
}
    if (curret -> next != NULL)
    {
        current -> next -> prev = curret -> prev
        }
            free (curret);
            return;
        }
    curret = curret -> next;
    }
    pf ("Node %d found", value);
    }
```

```
Void display (struct node * head)
{
    pf ("Doubly LL");
    while (head != NULL)
{
    pf ("%d", head -> data);
    head = head -> next;
}
    pf ("NULL");
}

int main()
{
    struct node * head = NULL;
    head = CreateNode (1);
    head -> next = CreateNode (2);
    head -> next -> prev = head;
    head -> next -> next = CreateNode(3);
    head -> next -> next -> prev = head
                                -> next;

display (head);

insertAft (head, head->next, 10);
    pf ("After insertion");
    display (head);

deleteNode (head, 2);
    pf ("After deletion");
    display (head);
    return 0;
}
```

Date __/__/__
Page _____

Output :

Doubly Linked List : 1 <-> 2 <-> 3 <-> NULL
After insertion :
Doubly LL : 1 <-> 10 <-> 2 <-> 3 <-> NULL

After deletion :
Doubly LL : 1 <-> 10 <-> 3 <-> NULL

11/2/1

Leetcode

```
#include <stdlib.h>
struct ListNode* splitListToparts (struct
ListNode* head, int K, int* returnSize)
{
    struct listnode* current = head;
        int length = 0;
    while ( current) {
            length ++;
        current = current -> next;
    }

    int part size = length/K;
    int extra nodes = length % K;
    struct Listnode** result = (struct listNode
    malloc (K* size of (struct listnode)));

        current = head;
        for (i= 0; i< K; i++)
{       struct listnode* part head = current
    int part length = part size +
    (i< extra nodes ? 1 : 0);
    for ( int j=0; j< part length -1 && extra
    nodes ? 1++)
{
        current = current.next;
    }
        if ( current){
            struct listNode* part head = curr
    int part length = part size + ( i< extra nodes
        1 : 0);
    for ( int j=0; j< part length -1 ; current;
                                        j++)
```

Date __/__/__
Page _____

```
else
    {   result[i] = NULL;
    }
    *return size = k;
    return result;
}
```

O/P:

[[1], [2], [3], [], []]

11/2/24

16/9/24

Binary Search Tree

Date __/__/__
Page ____

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

typedef struct *bst
{
        int data;
        struct bst * left;
        struct bst * right;
} node;

node* create ()
{
        node* temp;
        printf ("Enter data");
        temp = (node*) malloc (size of (node));
        scanf ("%d", temp -> data);
        return temp;
}

void insert ( node *root, node* temp)
{
        if ( temp -> data < root -> data)
        {
                if (root->lef != NULL)
                {
                        insert ( root->left, temp);
                }
        else
                {
                        root->left = temp;
                }
        }
```

29/2/24

Date __/__/__
Page _____

```
if (temp->data > root->data)
{
    if (root->right != NULL)
    {
        insert (root->right, temp)
    }
    else
    {
        root->right = temp;
    }
}
}

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

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

```
void postorder (node *root)
{
    if (root != null)
    {
        postorder ( root->left );
        postorder ( root->right );
        pf ("%d", root.data);
    }
}

int main ()
{   int ch;
    node * root = Null, *temp;

    do
    {   print ("Enter your choice");
        print ("1. Insert 2. preorder
                3. Inorder 4. postorder");

    switch (ch)
    {
        case 1: insert (root, temp);
                break;

        case 2: preorder (root);
                break;

        case 3: inorder (root);
                break;
```

```
Case 4:  postorder (root);
         break;

Case 5:  exit (0);
         break;
    }
}
    while (ch != 5)
        return 0;
}
```

Output :

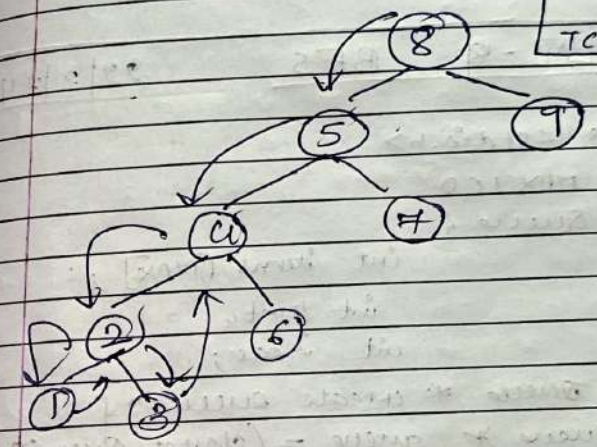
```
Enter your choice : 2
1. Insert
2 Preorder
3 Inorder
4 Postorder

Enter value to be inserted : 2

Enter choice : 3

1. Insert
2. Preorder
3. Inorder
4. Postorder

Enter choice : 4

Enter value to be inserted : 7
```

Leetcode

```
int GetLength ( struct ListNode* head)
{
    struct ListNode *temp = head;
    int count = 0;
    while (temp != NULL)
    {
        count++;
        temp = temp -> next;
    }
    return count;
}

struct Listnode * rotateRight ( struct
    Listnode * head, int K)
{
    int length = GetLength (head);
    struct Listnode *temp = head;
    struct Listnode *temp1;

    if (head == NULL ) return head;
elseif (head -> next == NULL)
    return head;

else
{   int i = length % K;
    while (i>0)
{
    temp = head;
    while (temp -> next = NULL)
{
    temp1 = temp;
    temp = temp -> next;
}
    temp1 -> next = temp -> next;
    temp -> next = head;
    head = temp;
}
```

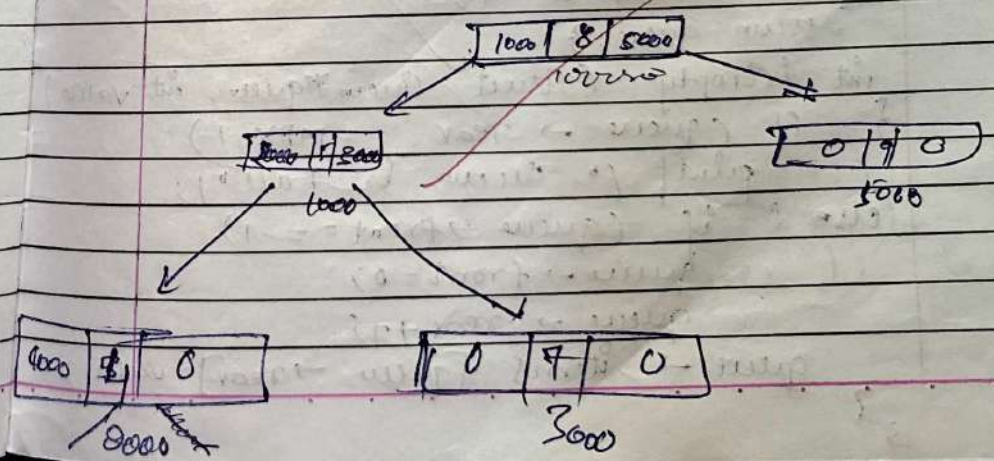
o/p:
Test Case1: [1,2,3,4,5,6,7]
K = 6
→ [5,1,2,3,4]
Test Case2:
head = [0,1,2]
K = 4
TC3: [2,0,1]

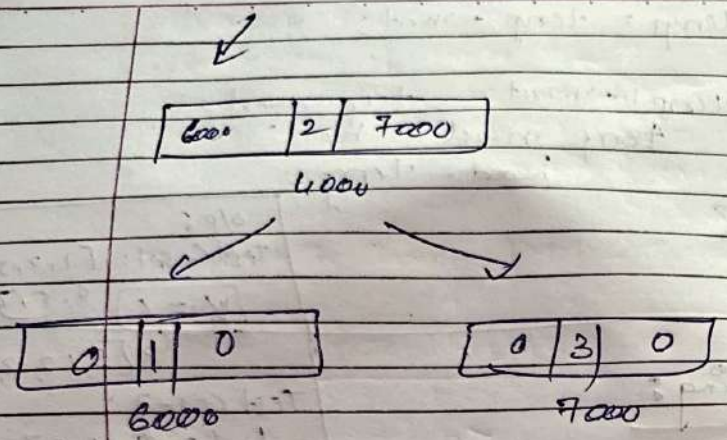
Tracing:



Preorder:

1 2 3 4 6 7 8 9





LAB - 9 BFS 22/2/24

```
#include <stdio.h>
#define MAX 100
struct Queue {
        int items [MAX];
        int front;
        int rear;
};
struct Queue * create Queue () {
struct Queue * queue = (struct Queue *)
malloc (sizeof (struct node()));
queue -> front = -1;
queue -> rear = -1;
return queue; }
int isEmpty (struct Queue *queue, int value)
{ if (queue -> rear == MAX-1)
      printf (" Queue is Full ");
else { if (queue -> front == -1)
        queue -> front = 0;
        queue -> rear ++;
  queue -> items [queue -> rear] = value;
}
```

```
int dequeue (struct queue * queue)
{ int item;
    if (isEmpty (queue))
printf (" Queue empty ");
    item = -1; } else {
item = queue -> items [queue -> front];
  queue -> front ++; } return item; }
int objMatrix [MAX] [MAX];
int visited [MAX];
void BFS (int start, int nodes) {
struct Queue* queue = create Queue ();
    visited [start] = 1;
enqueue [queue, start];
while (!isEmpty (queue)) { int currentNode
  = queue (queue);
printf ("current node");
for (int i = 0; i< nodes; i++) {
if (objMatrix [currentNode] [i] &&
    not visited [i])
{ visited [i] = 1;
  enqueue (queue, i); }
}
int main()
{ int nodes, startnode;
printf (" Enter no. of nodes");
scanf ("%d, &nodes");
printf ("Enter matrix");
for (i = 0; i< nodes; i++)
  for (j = 0; j< nodes; j++)
  { scanf ("%d", &[i][j]);
BFS (startnode, nodes); }
```

Date __/__/__
Page _____

Output :

Enter no. of vertices : 7

Enter matrix : 0101000
1011011
1011010
0001100
1100100

The graph is connected

DFS - LAB 9

```
#include <stdio.h>
#define MAX 10
void dfs (int graph [MAX vertices][MAX];
void numvertices (Bool visited [MAX];
int vertex {
        visited [(vertex)] = true ;
        int i;
  for ( int i=0 ; i< numVertices ; i++)
  {
    if ( graph (vertex [i]) = && ! v [i])
  dfs (graph, num , vertices , visited[i]);
  }
} bool isconnected (int graph (MAX]
{ bool visited [MAX] = false ;
  dfs (graph, num - vertices, visited, 0)
  int i;
for (int i=0; i< num_vertices ; i++)
{
    if (!visited [i]){
      int false;
    }
    return true;
} ;
```

```
int main ()
{
    int num = vertices;
    printf (" Enter vertices");
    scanf ("%d", &num_vertices) ;
      int i, j;
int graph [MAX] [MAX vertices]
printf (" Enter matrix ");
for ( i=0; i< mod_vertices ; i++)
{ for (j=0) j<mod_vertices ; j++)
{ scanf ("%d", graph [i] [j]);
}
} if (is connected (graph,num_vertices) {
    printf (" The graph is connected ");
else { printf (" Not connected ");
}
    return 0;
}
```

Output :

Enter no. of vertices : 7

Enter matrix :

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

Enter starting vertex : 2

2 1 3 4 5 0 6

10 Hashing - Linear Probing

```
#define Table size 100
#define key-length 5
#define Max name length 50
struct Employee {
  char key [key-length];
  char name [MAX-name-length];
  char designation [MAX-designation-length];
  float salary; };
struct Hashtable {
  struct Employee * table [Table-size];
};
int hash_function (const char *key, n)
{
  int sum = 0;
  for ( i=0; key[i] ; i++ ),
  {
      sum += key [i];
  }
    return sum ;
}
void insert (struct hash table, struct
    Employee *emp) {
  int index = hash - function (emp->key,
    Table-size);
  while ( ht->table [index] != NULL)
index = (index +1) % Table-size;
  ht -> table [index] != NULL)
  ht -> table [index] = emp;
}
struct Employee *search (struct hashtable
*ht, const char *key)
{
  int index = hash( Key, Table-size);
```

```
while (ht->table [index] ->key, key) == 0.
{ return ht-> table [index]
} index = (index +1) % Table-size;
} return null;
} int main ()
{ struct Hashtable ht;
  struct Employee *emp;
char key [key-length];
char filename [100];
char line [100];
for (i=0; i<Table-size ; i++) {
    ht.table[i] = null; }
printf (" Enter filename ");
scanf ( " %d, filename);
if (file == NULL) {
printf (" Error "); return 1; }
while ( line, sizeof (line), file)) {
emp = (struct Employee *) malloc (sizeof
    (struct employee));
insert ( &ht, emp);
}
int choice;
do { printf ("1. search 2. exit ");
printf (" &choice); switch (choice) {
case 1: search ();
        break;
case 2: printf (" Exit ");
        break;
: } while (choice != 2);
}
return 0; }
```

Output :

Item found at index 6.

2/3/24

23/2/24 Hacker Rank

Swapping Subtrees function:

```
void SwapAtLevel (node* root, int k, level) {
    if (root == NULL) return;
    if (level % k == 0) {
    node* temp = root->left;
    root->left = temp; }
swapAtLevel (root->left, k, level+1);
swapAtLevel (root->right, k, level+1); }
int ** swapNodes (int index_rows, int i)
colums, int ** indexes, queries)
Node ** node = (Node**) malloc (index+1)
                sizeof (node));
for (i=0; i<indexes_row; i++) {
  int leftindex = index [i][0];
  int rightindex = index [i][1];
if (leftindex != -1) nodes (i+1) -> left =
  nodes [leftindex]

if (rightIndex != -1) nodes(i+1) -> right
  = nodes (right -> root);

for (i=1; i<=n; i++)
{   if (S[i]) count ++;
if (count == n)
        printf ("connected");
  else  printf ("not connected");
    return 0;
}
```

Output: Enter vertices: 6

Enter matrix:

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

1 → 2
2 → 4
4 → 3
3 → 5
3 → 6

Connected