# INDEX

Name Nidhi. A Std Sec 3-D

Roll No. Subject Data Structures School/College

| Sl. No. | Date | Title | Page No. | Teacher Sign/ Remarks |
|---|---|---|---|---|
| 1. | 7/12/23 | Lab - 0 | 10 | |
| 2. | 21/12/23 | Lab - 1. | 10 | |
| 3. | 28/12/23 | Lab - 2 | 10 | |
| 4. | 11/1/24 | Lab - 3 + Leetcode Stack | 10 | |
| 5 | 18/1/24 | Lab - 4 + Leetcode Linked List | 10 | |
| 6 | 11/1/24 | Leetcode Stack | 5 | |
| 7 | 18/1/24 | Leetcode Linked List | 10 | |
| 8 | 25/1/24 | Lab 06 | 10 | |
| 9 | 1/2/24 | Lab 07 - Doubly Linked List | 10 | |
| 10. | 1/2/24 | Leetcode - Singly Linked List Splitting. | 10 | |
| 11. | 15/2/24 | Lab - 08 - Binary Search Tree | 10 | |
| 12. | 15/2/24 | Leetcode - Rotate Linked List | 10 | |
| 13. | 22/2/24 | Lab - 09 - BSF & DSF | 10 | |
| 14. | 22/2/24 | Hackerrank - Swap Node | 10 | |
| 15. | 29/2/24 | Lab - 10 - Hashing | 10 | |

10/23

# OUTPUT

1) Enter Username: At Nidhi
Enter Account number: 1234
Enter deposit = 9000
Account created successfuly

Enter amount to withdraw = 2000
Account created succesfuly.
Balance amount = 7000

2) Enter number of strings: 5
String 1: Ash
String 2: Tea
String 3: coffee
Sorted order: Ash Coffee Tea

3) Enter Array: 5 8 9 3 8 6 5 4 2 1
Enter key: 3
Key 3 is found at position 4

4) Enter string: Hattrick
Enter substring: trick
Substring trick is found at position 3

5) Enter array: 4 5 3 1 6 3 8 3 7 4
Enter key: 3
Last occurance of key is found at 8 position

6) Enter array: 4 5 3 1 6 3 8
Enter key: 1
Key 1 formed at 4 position

papergrid

Date: / /

(7) Enter array: 7 8 6 5 1 3 5 4 3
Enter key: 4
Key 4 is found at 8 position

(8) Enter array: 5 35 81 -6 51 20
Max element = 81
min element = -6

papergrid

Date: / /

21/12/23

# LAB-01

## I Swapping using pointers

```
#include<stdio.h>
void swap(int *a, int *b);
int ~~void~~ main() {
  int a, b;
  printf("Enter a & b\n");
  scanf("%d %d", &a, &b);
  printf("a=%d , b=%d", a, b);
  swap(&a, &b);
  printf("after swapping a=%d, b=%d", a, b);
  return 0;
}

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp; }
```

OUTPUT

```
Enter a & b
12  33
a = 12   b = 33
after swapping a = 33  b = 12.
```

## II Dynamic Memory Allocation

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

int main(){
 printf("Memory allocation by malloc\n");
 int *ptr = (int*) malloc(5*(sizeof(int)));
```

papergrid

Date: / /

```
if (ptr == NULL){
    printf(" memory allocation failed"); }

for (int i=0; i<5; i++){
    ptr[i] = i;
    printf(" %d \t", ptr[i]); }

printf(" \n Memory allocation by calloc \n")

{int * ptr1 = (int *) calloc (4, (sizeof (int)))
if (ptr1 == NULL){
    printf(" Memory allocation failed \n"); }
for (i=0; i<4; i++){
    ptr1[i] = i*2;
    printf (" %d \t", ptr1[i]); }

printf (" \n Memory allocation by realloc \n");

int * ptr2 = (int *) realloc (ptr1, 6 * sizeof (int));
if (ptr2 == NULL){
  printf (" Memory re-allocation failed \n");
    free (ptr1); }
for (i=4; i<6; i++){
    ptr2[i] = i*2; }
for (i=0; i<6; i++){
  printf (" %d \t", ptr2[i]); }

free (ptr2);
printf (" Memory after free \n");
for (i=0; i<6; i++){
    printf (" %d \t", ptr2[i]);
```

```
printf ("Memory allocation through calloc");
ptr2 = (int *)calloc (4, sizeof(int));
for (int i=0; i<4; i++);{
   ptr2[i]= i*2;
   printf("%d/t", ptr2[i]); }
return 0;
}
```

OUTPUT

Memory allocation through malloc
0 1 2 3 4
through calloc
0 2 4 6
Memory allocation through realloc.
0 2 4 6 8 10
Memory ~~realloc~~ after free
1324400 0 1328608 0 0 10
Memory re-allocation through calloc
0 2 4 6

III

## Stack operations

```
# include <stdio.h>
# include <stdlib.h>
# define max 100
int top = -1;
int stack[max];
void push (int a);
int pop()
void display();
```

papergrid

Date: / /

```
int main(){
    int arr[100],size;
    printf("Enter array size:");
    scanf("%d", &size);
    printf("Enter values of stack\n");

    for (int i=0; i<size; i++){
        scanf("%d", &arr[i]);
        push (arr[i]); }
    printf("Display stack before popping\n");
    display();

    For (i=size-1; i>=0; i--){
        pop(); }
    printf("stack after popping \n");
    display();
    return 0; }


void push (int a){
    if (top == max-1){
        printf("stack overflow\n");
        return; }
    top = top + 1
    stack[top] = a; }

int pop(){
    if (top == -1){
        printf("stack underflow\n");
        return -1; }
    top--;
    return stack[top]; }
```

papergrid

Date: / /

```
void display(){
    if (top == -1){
        printf("Stack empty\n");
        return ;}
  printf("Stack elements:\n");
  for (int i=0; i<=top; i++){
      printf("%d\t", stack[i]);}
  printf("\n");
}
```

OUTPUT

Array size : 7
Enter values of stack:
1 2 3 4 5 6 7
Stack before popping
Stack elements:
1 2 3 4 5 6 7
After popping
Stack elements:
Array is empty.

21/12/23

28/02/23

LAB-02

# Postfix from Infix

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#define max 100

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

void push (char item){
    if(top==max-1){
      printf("Stackoverflow\n");
      exit(0); }
    else { top++;
      stack[top] = item; }
    }

char pop(){
    char item;
    if (top==-1){
        printf("Stack underflow\n");
        exit(1); }
    else { item= stack[top];
        top--;
        return item; }

int precedence (char sym){
    if (sym=='^')
        return 3;
    else if (sym=='*' || sym=='/')
        return 2;
    else if (sym=='+' || sym=='-')
        return 1;
    else
        return 0;  }
```

```
int    isOperator (char sym){
    if(sym=='^'||sym=='*'||sym=='/'||sym=='+'
        ||sym=='-')
        return 1;
    else
        return 0;}

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

    while (infix[i] != '\0'){
        sym = infix[i];
        if (sym == '(')
            push(sym);
        else if (isalnum(sym))
            postfix[j++] = sym;
        else if (sym == ')'){
            while (stack[top] != '(' ){
                temp = pop();
                postfix [j++] = temp; }
            pop(); }
        else if (isOperator(sym)){
            while (precendence(stack[top]) >= precedence(sym))
            {   temp = pop();
                postfix[j++] = temp; }
            push (sym); }
        i++;
    }
    while (stack[top] != '#'){
        temp = pop();
        postfix[j++] = temp;}
    postfix[j] = '\0'; }
```

papergrid
Date: / /

```
int main() {
    char infix[max], postfix[max];
    printf("Enter infix expression : ");
    gets(infix);

    infixToPostfix(infix, postfix);
    printf("Postfix expression: %s \n", postfix);
    return 0;
}
```

OUTPUT

Enter infix expression: (1*7-9+6)/6^2
Postfix expression: 17*9-6+62^/

2b) Postfix Evaluation

```
#include <stdio.h>
#include <stdlib.h>
# define size 50

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

void push(int item) {
    if (top == size - 1){
        printf("Stack overflow \n");
        exit(0); }
    else { top++;
        stack [top] = item; }
}
```

papergrid

Date: / /

```
int pop () {
    int item;
    if (top == -1) {
        printf("Stack underflow\n");
        exit(1); }
    else { item = stack [top];
        top = -;
        return item; }
}

int compute (char op, int op1, int op2){
    switch (op){
    case '+':
        return op1 + op2;
    case '-':
        return op2 - op2;
    case '*':
        return op1 * op2;
    case '/':
        if (op2 != 0){
        return op1 / op2; }
        else {
        printf ("Error: Div by zero\n");
        exit(1); }
    default: printf ("invalid Exp"); }
}

int evalPost(char exp[]) {
    for (int i=0; exp[i] != '\0'; i++){
        if (isdigit(exp[i])){
            push(exp[i] - '0'); }
    else if (exp[i]=='+' || exp[i]=='-' || exp[i]==
            '*' || exp[i]=='/') {
        int op2 = pop();
        int op1 = pop();
```

```
        int result = compute(exp[i], op1, op2);
        push(result);
    }
}
return stack[top];
}

int main(){
    char postExp[size];
    printf("Enter postfix expression: ");
    gets(postExp);
    int result = evalPost(postExp);
    printf("Result : %d \n", result);
    return 0;
    }
```

Output

Enter Postfix evaluation: 17*9-6+62^/
Result : 3

Swaha
28/12/23

papergrid

Date: / /

11/1/24

Lab 03

Queue Implementation

```
#include<stdio.h>
#include <stdlib.h>
# define max 5
int front=0;
int rear = -1;
int q[max];

void enqueue( int elem){
    if ( rear = = max-1){
        printf(" Queue full \n");
        return ;}
    rear ++;
    q[rear] = elem; }

void display(){
    for(int i=front ; i<=rear ; i++){
        printf("%d \t", q[i]); }
    }

int dequeue () {
    if ( rear < front){
        printf(" Queue empty \n");
        return 0; }
    front ++ ;
    return q[front]; }

int main () {
    int ch, i, elem;
    while (1){
    printf(" 1.INSERT \n 2.DELETE \n3.DISPLAY
        \n 4. EXIT \n");
    scanf(" %d ", &ch);
```

```
switch (ch){
    case 1 : printf("Enter value to be entered
             scanf("%d", &elem);
             enqueue(elem); break;
    case 2 : dequeue(); break;
    case 3 : display(); break;
    case 4 : exit(0);
    default : printf("Enter valid input\n");
              exit(0);
    }
  }
}
```

Output

```
1. Insert
2. Delete
3. Display
4. Exit.
1
Enter element to be entered : 2
1. Insert
2. Delete
3 Display
4. Exit
1
Enter element to be entered : 4
1. Insert
2. Delete
3. Display
4. Exit
2.
1. Insert
2. Delete
3. Display
4. Exit
```

3.

4

## Circular Queue

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

int q[size], F=0, r=-1;
int count = 0;

void enqueue (int item){
    if (count == size){
        printf("\n Queue full!");
        return; }
    q[++r] % size = item;
    count++; }

void dequeue () {
    if (count == 0){
        printf ("\n Queue empty");
        return; }
    f = (f+1) % size;
    count--; }

void display () {
    if (count == 0){
        printf ("\n Queue empty");
        return; }
    int front = f;
    f = (f+1) % size;
    count--; }
```

papergrid
Date: / /

```
int main () {
        int ch, item;
        while (1) {
            printf (" \n Select choice 1. Enqueue 2. Dequeue
        ~~scanf~~ 3. Display: ");
            ~~ch~~ switch(ch) {
            case 1:
                printf (" \n Enter value to insert: ");
                scanf ("%d", &item); ~~break~~
                enqueue (item); break;
            case 2:
                ~~pop~~ dequeue ();
                printf ("\n Item popped "); break;
            case 3:
                display (); break;
            default: exit(0); }
        }
}
```

Select choice 1. Enqueue 2. Dequeue 3. Display
1
Enter value to insert 5
Select choice 1. Enqueue 2. Deque 3. Display
1
Enter value to insert 4
Select ~~value~~ choice ~~to~~ 1. ~~insert~~ Enqueue 2. Dequeue
3. Display
~~Dequeue~~ 2
Select choice 1. Enqueue 2. Dequeue 3. Display
3
4.

papergrid

Date: / /

# LINKED LIST

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

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

struct node * head = NULL;
void display () {
    struct node *ptr = head;
    if (ptr = = NULL) {
        printf (" list is empty \n");
        return; }
    printf (" Element are: ");
    while (ptr ! = NULL) {
        printf("%d", ptr -> data);
        ptr = ptr -> next; }
        printf ("\n"); }

void insert_begin () {
    struct node *temp;
    temp = (struct node *) malloc (sizeof (struct node))
    printf (" Enter the value to be inserted: ");
    scanf ("%d", & temp -> data);
    temp -> next = head;
    head = temp; }

void insert_end () {
    struct node * temp; * ptr;
    temp = ( struct node *) malloc (sizeof (struct
    *           node)) ;
    printf ("%d", & temp -> data);
    temp -> next = NULL;
```

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

void    insert_pos () {
        int     pos, i;
        struct node *temp, *ptr;
        temp = (struct node *) malloc (sizeof (struct node
        printf ("Enter the position to insert:");
        scanf ("%d", &pos);
        printf ("Enter the value to be inserted:
        scanf ("%d", &temp->data);
        temp-> next = NULL;

        if (pos ==0) {
            temp -> next= head;
            head = temp;
        else {
            ptr = head;
            for (i=0; i<pos-1; i++){
                ptr = ptr -> next;
                if (ptr == NULL){
                    printf ("Position not found \n");
                    return; }
            }
        temp -> next = ptr -> next;
        ptr -> next = temp;
        }
}
```

papergrid

Date: / /

```
int main () {
    int choice;
    ~~choice~~ while (1) {
        printf (" \n 1-Insert at the beginning
        \n 2. Insert at the end \n 3. Insert
        at any position \n 4. Display
        \n 5. Exit \n");
        printf (" Enter your choice: ");
        scanf (" %d ", &choice);

        switch (choice) {
            case 1: insert_begin (); break;
            case 2: insert_End (); break;
            case 3: insert_pos (); break;
            case 4: display (); break;
            case 5: exit(0); break;
            default: printf(" Enter the correct choice);
        }
    }
return 0;
}
```

OUTPUT

```
1. Insert at the beginning.
2. Insert at the end
3. Insert at the any position
4. Display
5. Exit
Enter your choice : 1
Enter the value to be inserted : 12
```

papergrid
Date: / /

1. Insert at the beginning
2. Insert at the end
3. Insert at any ~~site~~ position
4. Display
5. Exit.
Enter your choice : 4
Enter the value to be inserted : 12

1. Insert at the beginning
2. Insert at the end
3. Insert at any position
4. Display
5. Exit.
Enter your choice : 2
Enter the value to be inserted : 56

1. Insert the beginning
2. Insert at the end
3. Insert at any position
4. Display
5. Exit
Enter your choice : 3
Enter the position to insert : 2
Enter the value to be inserted : 20

1. Insert the beginning
2. Insert at the end
3. Insert at any position
4. Display
5. Exit
Enter your choice : 4
Elements are : 12 56 20

papergrid

Date: / /

1. Insert at beginning
2. Insert at the end
3. Insert at any position
4. Display
5. Exit.

Enter your choice

5

11/1/24

Leetcode 1 - Stack

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

typedef struct {
        int val;
        int min; } Node;

typedef struct {
        Node* stack;
        int top, capacity; } Minstack;

Minstack * minstackCreate () {
   Minstack * stack = (Minstack*) malloc (sizeof (Minstack));
   stack -> stack = (Node*) malloc (sizeof (Node) * 100);
   stack -> top = -1;
   stack -> capacity = 100;
      return stack; }
```

papergrid

Date: / /

```
void minStackPush (Minstack * obj, int val){
    if (stack -> top == -1) {
        obj -> stack [++(obj -> top)].val = val;
        obj -> stack [obj -> top]. min = val; }
    else {
        obj -> stack [++(obj -> top)]. val = val;
        obj -> stack [obj -> top]. min = (val < obj -> stack[
                obj -> top - 1]. min)? val : obj -> stack[
                    obj -> top - 1]. min; }
}

void minStackPop (MinStack * obj){
    if (obj -> top >= 0){
        obj -> top - - ; }
    }

int minStackTop (MinStack * obj){
    return obj -> stack [obj -> top]. val ; }

int minStackGetMin (MinStack * obj){
    return obj -> stack [obj -> top]. min ; }

void minStackFree (Minstack * obj) {
        free (obj -> stack);
            free (obj);
    }
```

~~OUTPUT~~ INPUT

["Minstack", "push", "push", "push", "GetMin", "pop", "top", "getMin"]

[[], [-2], [0], [-3], [], [], [], []]

papergrid

Date: / /

OUTPUT

[null, null, null, null, -3, null, 0, -2].

18/1/24 Lab - 04 - Singly linked List

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

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

struct node * head = NULL;

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

void insert_begin () {
    struct node *temp = (struct node*) malloc
                (sizeof (struct node));
    printf ("Enter the value to be inserted:
    scanf ("%d", &temp->data);
    temp-> next = head;
    head -> temp; }

void delete_end () {
    if (head == NULL){
      printf ("List is empty. Deletion is not
                possible.");
      return; }

    struct node * *temp = head;
    head = head -> next;
    printf ("Element deleted from the beginning
            : %d \n", temp->data);
```

```
    free(temp); }

void delete_end() {
    if (head == NULL){
        printf("list is empty \n");
        return; }
    struct node *temp, *prev;
    temp = head;
    while (temp -> next != NULL){
        prev = temp;
        temp = temp -> next; }
    if (temp == head){
        head = NULL; }
    else{
        prev -> next = NULL; }
    printf("Element deleted from the end: %d\n",
            temp -> data);
    free(temp); }

void delete_at_position() {
    int position;
    printf("Enter the position to delete: ");
    scanf("%d", &position);
    if (head == NULL){
        printf("list is empty \n");
        return; }
    struct node *temp, *prev;
    temp = head;
    if (position == 0){
    head = head -> next;
    printf("Element at position %d deleted
            successfully \n", position);
    free(temp);
    return; }
```

```
for (int i=0; temp != NULL && i < position, i++)
        prev = temp;
        temp = temp -> next; }

if (temp == NULL) {
    printf (" Position %d is out of bounds\n",
            position);
        return; }
prev -> next = temp -> next;
printf (" Element at position %d deleted",
        position);
free (temp);
}


int main() {
    int choice;
    while (1) {
    printf ("1. to insert at the beginning
            \n 2. to delete from beginning
            \n 3. to delete from end
            \n 4. to delete from any position
            \n 5. display
            \n 6. exit \n");
    scanf ("%d", &choice);
    switch (choice) {
    case 1: insert_begin(); break;
    case 2: delete_begin(); break;
    case 3: delete_end(); break;
    case 4: delete_at_position(); break;
    case 5: display(); break;
    case 6: exit(0); break;
    default: printf (" Enter correct choice");
            break; }
} return 0; }
```

papergrid

Date: / /

# OUTPUT

1. To insert at the beginning
2. To ~~insert~~ delete from beginning
3. To delete at end
4. To delete from any position
5. To display
6. Exit

Enter choice: 1
Enter value to be inserted: 1

Enter choice : 1
Enter value to be inserted: 2

Enter choice : 1
Enter value to be inserted: 3

Enter choice: 1
Enter value to be inserted: 4

Enter choice : 1
Enter value to be inserted: 5

Enter your choice: 2
Element deleted from beginning : 5

Enter choice : 3
Element deleted from end : 1

Enter your choice: 4
Enter position to delete : 2
Element at position 2 deleted successfully

Enter choice : 5
Elements are : 4 → 3 → NULL

Enter choice : 6
Exit

LEETCODE - Linked List

```
struct ListNode* reverseBetween ( struct
    listNode * start, int a, int b){
    a = 1;
    b = 1;
    struct listNode * node1 = NULL, *nodea=NULL,
    *ptr = start, * node2 = NULL, nodeb* = NULL,
    int c = 0;
    while (ptr != NULL){
        if (c == a-1)
            nodeb = ptr;
        else if ( c == a){
            node1 = ptr;
        else if ( c == b)
            node2 = ptr;
        else if (c == b+1){
            nodea = ptr; break;}
        c += 1;
        ptr = ptr -> next; }

struct ListNode * pre = nodea, *temp;
    ptr = start;
    c = 0;
```

papergrid

Date: / /

```
while( ptr != NULL){
    if (c>=a && c<b){
        temp = ptr -> next;
        ptr -> next = pre;
        pre = ptr;
        ptr = temp; }
else if (c==b){
    ptr -> next= pre;
    if (a==0)
        start = pts;
    else
        notleb -> next = pts;
    break;
}
else
    pts = ptr -> next;
    c+=1; } return start; }
```

INPUT

head = {1, 2, 3, 4, 5}
left = 2
right = 4

OUTPUT
{1, 4, 3, 2, 5}

INPUT
head = {5}
left = {1}
right 1

OUTPUT
{5}

18/1/24

papergrid

Date: / /

25/1/24

# Lab - 06

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

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

struct Node *createNode (int value){
    struct Node* newNode = (struct node*) malloc
            (sizeof (struct Node));
    newNode -> data = value;
    newNode -> next = NULL;
    return newNode; } ;

void insertEnd (struct Node* head, int value){
    struct node * newnode = createNode (value);
    if ( *head = = NULL) {
        * head = newNode ; }
    else {
        struct node* temp = *head ;
        while ( temp -> next ! = NULL) {
            temp = temp -> next; }
        temp -> next = newNode ; } }

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

papergrid

Date: / /

```
void sortLinkedlist (struct node* head){
    int swapped, i;
    struct node * *ptr,
    Struct node* lptr = NULL;
    if (head == NULL)
            return 0;
    do{ swapped = 0;
        ptr = head;
        while (ptr -> next != lptr){
            if (ptr->data > ptr->next->data){
                int temp = ptr -> data;
                ptr -> data = ptr -> next -> data;
                ptr -> next -> data = temp;
                swapped 1;
            }
            ptr = ptr -> next; }
        lptr = ptr; } while (swapped); }

void node* concatenate Linked list (struct node* list1,
    struct node* list2){
    if (*list1 == NULL){
        *list1 = *list2; }
    else{ struct node* temp = *list1;
        while (temp -> next != NULL){
            temp = temp -> next; }
        temp -> next = list2; } }

struct node* reverse list (struct node* head){
    struct node * prev = NULL, *current = head,
    *next = NULL;
    while (current != NULL){
        next = current -> next;
        current -> next = prev;
        prev = current;
        current = next; }
```

papergrid
Date: / /

```
return prev; }

int main(){
    struct node* l1 = NULL;
    struct node* l2 = NULL;
    int n, value;
    printf("Enter no. of elements for L1");
    scanf("%d", &n);
    printf("Enter ~~no. of~~ elements for l1");
    for (int i = 0; i < n; i++){
        scanf("%d", &value);
        insertEnd(&l1, value); }
    printf("Enter elements for l2");
    for (int i = 0; i < n; i++){
        Scanf("%d", &value);
        insertEnd(&l2, value); }
    sortedLinkedlist(list1);
    printf("Sorted L1:");
    display(L1);
    l2 = reverselinkedlist(l2);
    printf("Reversed L2:");
    display(l2);
    concatenatelists(&L1, &l2);
    printf("Concatenated List:");
    display(l1);
    struct node* temp;
    while(l1 != NULL){
        temp = l1;
        l1 = l1 -> next;
        free(temp); }
    return 0; }
```

papergrid

Date: / /

OUTPUT

Enter the number of elements list1: 5
Enter the element of L1: 2 6 3 4 2
Enter the number of element list2: 4
Enter the elements: 12 14 13 15

Sorted L1 = 2 → 2 → 3 → 4 → 6 → NULL
Reversed list L2 = 15 → 13 → 14 → 12 → NULL
Concanted list: 2 → 2 → 3 → 4 → 6 → 15 → 13 → 14 → 12 → NULL

## Stack

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

struct node* createNode (int value) {
  struct node * newNode = (struct *node)malloc(
      sizeof (struct node));
    newdata → data = ~~no~~.value;
    newNode → next = NULL;
    return newNode; }

void push (struct node** top, int value){
    struct node* newNode = createNode(value);
      newNode → next = *top;
      *top = newNode; }

int pop( struct node ** top){
    if (*top == NULL){
        printf(" Stack Underflow \n"); }
```

papergrid

Date: / /

```
struct node* temp = *top;
int poppedValue = temp->data;
*top = temp->next;
free(temp);
return poppedValue;}

int main(){
struct top* = NULL;
int choice, value;
do {
printf("\n Stack Op");
printf("1. Push \n 2. Pop \n 3. Display \n 4.
        Exit \n");
printf("Enter your choice");
scanf("%d", &choice);
switch(choice)
case 1: push(&top, value); break;
case 2: pop(&top); break;
case 3: display stack(top); break;
case 4: exit(0); break;
default: printf("Enter valid choice");}
} while(choice != 4);
while (top != NULL){
temp = top;
top = top->next;
free(temp);
}
return 0;
}
```

OUTPUT

Stack operation

1. Push

2. Pop
3. Display
4. Exit
Enter choice = 1
Value to push = 3
~~Stack op~~ (options)
Enter choice = 1
Value to push = 2
(options
Enter choice : 1
Value to push = 1
Options
Enter choice : 2
Value Popped = 1
Enter choice : 3
Stack = 3 2 .

## Queue using Singly linked list

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

struct Queue {
    struct node *front;
    struct node* rear; };

struct node* createNode(int value){
    struct node* newNode = (struct node*)malloc
        (sizeof(struct node));
```

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

struct Queue * createQueue() {
    struct Queue * queue = (struct Queue*) malloc
        (sizeof(struct Queue));
    queue -> front = queue -> rear = NULL;
    return queue; }

void enqueue(struct Q* q, int value){
    struct node* newNode = createNode(value);
    if (queue -> rear = NULL){
        queue -> front = queue -> rear = newNode;
            return;}
    queue -> rear -> next = newNode;
    queue -> rear = newNode; }

int dequeue(struct q* q) {
    if (que -> front == NULL)
        printf("Queue underflow \n");
    struct node* temp = que -> front;
    int dequeueVal = temp -> data;
    q -> front = temp -> next;
    if (q -> front == NULL) {
        q -> rear = NULL; }
    free(temp);
    return dequeueValue; };

int main() {
    struct q* q = createQueue();
    int choice, value;
    do{
```

```
printf("\n Queue Op");
printf("\n 1. Enqueue \n 2. Dequeue \n 3. Display
        \n 4. Exit");
printf("Enter choice ");
scanf("%d", &choice);
switch (choice){
case 1: enqueue(queue, value); break;
case 2: dequeue(queue); break;
case 3: displayQueue(queue);
case 4: printf("Exiting");
        exit(0); break;
default: printf("Enter valid opt");
}
} while (choice != 4);
struct node* temp;
while (queue->front != NULL) {
    temp = q->front;
    q->front = queue->front->next;
        free(temp);
}
free(queue);
return 0;
}
```

## OUTPUT

```
Queue operations
1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your choice = 1
Enter value to insert = 2
```

papergrid
Date: / /

(Options)
Enter your choice : 1
Enter value : 5
(Options)
Enter your choice : 2
Dequed value : 2
(Options)
Enter your choice : 1
Enter value to insert : 6
(Options)
Enter your choice : 3
Queue : 6 5 .
(Options)
Enter your choice : 4
Exiting the program.

Sneha
25/11/24

papergrid

Date: / /

# Lab -07 Doubly Linked List.

04/1/24

① WAP to implement doubly linked list with primitive operations

a) Create a doubly linked list
b) Insert a new node to the left
c) Delete the node based on specific value
d) Display contents.

```
#include <stdio.h>
#include <stdlib.h>
struct node {
        int data;
        struct node *next;
        struct node *prev; };
struct node *sl = NULL;
struct node * createNode (int value) {
    struct node *temp = (struct node *) malloc
                        (sizeof (struct node));
    temp -> data = value;
    temp -> next = NULL;
    temp -> prev = NULL;
    return temp; }

struct node *insert_left (struct node *start) {
    int value, key;
    struct node *temp = createNode (0);
    printf ("Enter the value to the left of which
            the node has to be inserted:");
    scanf ("%d", &key);
    printf ("Enter the value to be inserted");
    scanf ("%d", & temp -> data);
    struct node *ptr = start;
    while (ptr != NULL && ptr -> data != key) {
        ptr = ptr -> next; }
```

# Lab-07 Doubly Linked List

① WAP to implement doubly linked list with primitive operations
a) Create a doubly linked list
b) Insert a new node to the left
c) Delete the node based on specific value
d) Display contents

```
#include <stdio.h>
#include <stdlib.h>
struct node {
        int data;
        struct node *next,
        struct node *prev; };
struct node *s1 = NULL;
struct node * createNode (int value) {
    struct node *temp = (struct node *) malloc
                        (sizeof (struct node));

    temp -> data = value;
    temp -> next = NULL;
    temp -> prev = NULL;
    return temp; }

struct node *insert_left (struct node *start) {
    int value, key;
    struct node *temp = createNode (0);
    printf ("Enter the value to the left of which
            the node has to be inserted");
    scanf ("%d", &key);
    printf ("Enter the value to be inserted");
    scanf ("%d", &temp->data);
    struct node *ptr = start;
    while (ptr != NULL && ptr->data != key) {
        ptr = ptr -> next; }
```

papergrid
Date: / /

```
if (ptr == NULL) {
    printf (" Node with value %d not found \n", key);
    free(temp);}
else {
    temp -> next = ptr;
    temp -> prev = ptr -> prev;
    if (ptr -> prev != NULL){
        ptr -> prev -> next = temp; }
    ptr -> prev = temp;
    if (ptr == start) {
        start = temp; } }
return start; }

struct node * delete_value (struct node * start){
    int value;
    printf (" Enter val to delete");
    scanf ("%d", & value);
    struct node *ptr = start;
    while (ptr != NULL & ptr->data != value){
        ptr = ptr -> Next }
    if (ptr == NULL){
        printf (" Node not found") }
    else {
        start = ptr -> next; }
    if (ptr -> next != NULL){
        ptr -> next -> prev = ptr -> prev; }
    printf (" Node with %d deleted", value);
    free (ptr); }
    return start; } }
```

papergrid

Date: / /

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

struct node *insert_right (struct node *start){
    int value;
    struct node * temp = createNode (0);
    printf ("Enter the value to be inserted:),
    scanf (" %d", &temp -> data);
    if (start == NULL){
        start = temp; }
    else {
        struct node *ptr = start;
        while (ptr -> next != NULL){
            ptr = ptr -> next; }
        ptr -> next = temp;
        temp -> prev = ptr; }
    return start; }

int main() {
    int choice;
    while (1){
    printf (" \n 1. Create a dl list \n 2. Insert to
left \n 3. Delete based on val \n 4. Display
\n 5. ~~Dele~~ Insert to right \n
    switch (choice){
        case 1: sl = createNode(0);
                printf (Doubly list created"); break;
```

```
case 2: s1 = insert_left (s1); break;
case 3: s1 = delete_value (s1); break;
case 4: display (s1); break;
case 5: s1 = insert_right (s1); break;
case 6: printf ("Exiting");
        exit (0);
default: printf ("Invalid choice \n"); } }
return 0; }
```

## OUTPUT

1. Create a doubly linked list
2. Insert to left of a node
3. Delete based on a specific value
4. Display the content
5. Insert to right
6. Exit.

```
1
Doubly linked list created

<options>
5
Enter value to be inserted: 12
<options>
5
Enter value to be inserted: 13
<options>
2
Enter value to be inserted: 6
Enter value to the left of which
new node to be inserted: 13
<options>
5
Enter value to be inserted: 14
```

<options>
4
list contents
0
12
6
13
14
<options>
3
list contents: Enter the value to be deleted
13
Node with value 13 deleted.
<options>
4
list contents: 0 12 6 14

LEETCODE
on
Singly linked list

```
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 * sizeof(struct listNode*));
```

```
current = head;

for (int i = 0; i < k; i++){
    struct ListNode * part_head = current;
    int part_length = part_size + (i < extranodes ? 1 : 0);
    for (int j = 0; j < part_length - 1 && current; j++) {
        current = current -> next; }
    if (current) {
        struct ListNode ** next_node = current->
        current -> next = NULL;
        result [i] = part_head;
        current = next_node; }
    else { result [i] = NULL; }
  }
* returnSize = k
return result; }
```

OUTPUT

Case 1
head = [1, 2, 3]
k = 5.
Output = [[1], [2], [3], [], []]

Case 2: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
k = 3
Output = {[1, 2, 3, 4], [5, 6, 7], [8, 9, 10]}

1/2/24

papergrid

Date: / /

Lab - 08 - Trees

WAP i) To construct a binary Search Tree
ii) To traverse the tree using all methods
ie: inorder, preorder, postorder
iii) To display elements in tree

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

struct TreeNode {
    int data;
    struct TreeNode * left;
    struct TreeNode * right; };

struct TreeNode* create node (int data){
    struct TreeNode* newNode = (struct TreeNode*)
    malloc (sizeof (struct TreeNode));
    newNode -> data = data;
    newNode -> left = newNode -> right = NULL;
    return newNode; };

struct TreeNode * insertNode (struct Treenode* root,
                                int data) {
    if (root = = NULL){
        return createNode (data); }
    elseif ( data < root -> data){
        root -> left = insertNode (root -> left, data); }
    else if ( data > root -> data){
        root -> right = insertNode (root -> right, data); }
    return root; }

void inOrderTraversal ( struct TreeNode * root ){
    if ( root != NULL){
        inorderTraversal (root -> left);
        printf ("%d", root -> data);
        inOrderTraversal (root -> right); } }
```

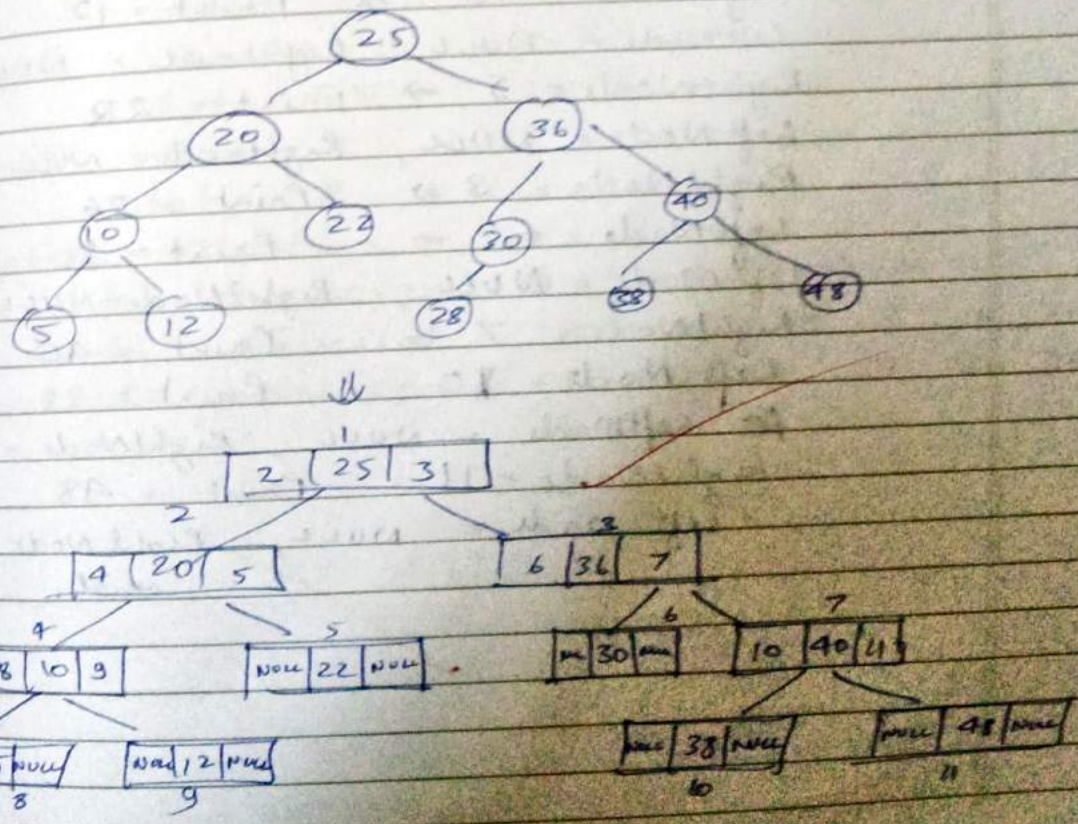
papergrid

Date: / /

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

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

void    displayTree ( struct TreeNode* root ){
        printf (" Inorder tree: ");
        inOrderTraversal (root);
        printf (" \n ")
        printf (" Postorder tree ");
        postOrderTraversal (root);
        printf ("\n ");
        printf (" Preorder tree ");
        preOrderTraversal (root);
        printf ("\n "); }

int     main () {
    struct TreeNode* root = NULL
    int choice, data;
    do {
        printf ("\n. 1. Insert a tree \n 2. Display a tree
                \n 3. Exit \n Enter your choice:")
        scanf (" %d", & choice);
        switch (choice){
```

```
case 1: printf (" Enter data to insert"),
        scanf ("%d", &data),
        root = insertNode (root, data),
        break;
case 2: if (root == NULL){
            printf (" Tree is empty \n"); }
        else { displayTree (root); }; break
case 3: printf (" Exiting program"), break
default: printf (" Invalid choice!");
        }
} while (choice != 3),
return (0);
}
```

Tracing



inorder = 5 10 12 20 22 25 23
30 36 38 40 48

Preorder = 25 20 10 5 12 22 36 30 28 38

Post Order = 5 12 10 22 20 28 30 48 38
36 25

Preorder:

root = 1 != NULL
∴ Print 25

Left Node = 2 Print = 20
Leftode = 4 Print = 10
Left node = 8 Print = 5.
Leftnode = NULL ⇒ recursion.
RightNode = 9 ⇒ Print = 12
LeftNode = NULL, Rightnode = NULL recurse
RightNode = 5 → Print = 22
LeftNode = NULL, Rightnode = NULL recurse;
RightNode = 3 ⇒ Print = 36
LeftNode = 6 = Print = 30
LeftNode = NULL, RightNode = NULL recurse;
RightNode = 7 = Print = 40
Left Node = 10 Print = 38
LeftNode = NULL, RightNode = NULL; recurse
RightNode = 11 Print = 48
Left Node = NULL, Right Node = NULL
end.

# OUTPUT

Enter
1. Insert a node
2. Display tree
3. Exit
Enter choices : 1
Enter data : 25
<opt>
Enter choices : 1
Enter data : 20
<opt>
Enter choice = 1
data = 10
<opt>
choices : 1
data : 12
<opt> choices : 1
data : 5
<opt> choice : 1
data : 8
<opt> choice : 2
In-order traversal = 5 8 10 12 20 25
Pre-order traversal = ~~25~~ 20 10 5 8 12
Post-order traversal = 8 5 12 10 20 25

papergrid
Date: / /

# heat code - Rotate Linked List

```
struct listNode* rotateRight( struct listNode* head,
    int k) {
  if (head == NULL || k == 0) {
        return head; }
  struct listNode* current = head;
    int length = 1;
    while (current -> next != NULL){
          current = current -> next;
          length++; }
    k = k % length;
     if (k==0) {
         return head;
    current = head;
    if( int i<1; i < length -k; i++) {
        current = current -> next; }
    struct listNode* newhead = current -> next;
     current -> next = NULL;
     current = newHead;
     while (current -> next != NULL){
          current = current -> next; }
        current -> next = head;
        return newHead; }
```

OUTPUT

head = {1, 2, 3, 4, 5}  k = 2

Output = {4, 5, 1, 2, 3}
Expected = {4, 5, 1, 2, 3}

15/2/24

papergrid

Date: / /

Lab - 09

Write a Program to traverse a graph using breadth first ~~transit~~ Search method.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdio.h>
# define MAX 50

typedef struct Graph_t {
    int v;
    bool adj[MAX][MAX]; } Graph;

Graph* Graph_create (int v){
    Graph* g = malloc (sizeof(Graph));
    g->v = v;
    for (int i = 0; i<v ; i++){
        for (int j = 0; j < v ; j++){
            g -> adj [i][j] = false; }
    }
    return g; }

void Graph_destroy (Graph* g){
    free(g); }

void Graph - addEdge (Graph * g, int v, int w){
    g -> adj[v][w] = true; }

void Graph_BFS (Graph* g, int s){
    bool visited [MAX];
    for (int i=0; i < g->v; i++){
        visited [i] = false; }
    int queue [MAX];
    int front = 0, rear = 0;
```

```
    visited [s] = true;
    queue [rear++] = s;
  while ( front != rear ){
        s = queue [front++];
        printf ("%d ", s);
        for (int adjacent = 0; adjacent < g->v; adjacent++)
            if (g->adj [s][adjacent] && ! visited [adjacent]){
                visited [adjacent] = true;
                queue [rear++] = adjacent; } } }

int main () {
    int numVertices;
    printf ("Enter the number of vertices in the graph:
    scanf ("%d", &numVertices);
    Graph* g = Graph_create (numVertices);
    int numEdges;
    printf ("\n Enter the no. of edges in the graph:");
    scanf ("%d", &numEdges);
    printf ("Enter the edges (vertex1 vertex2): \n");
    for (int i = 0; i < numEdges; i++) {
        int v, w;
        scanf ("%d%d", &v, &w);
        GraphEdge (g, v, w); }
  int startvertex;
  printf ("Enter start vertex for BFS:");
  scanf ("%d", &startvertex);
  printf ("Following is BFS (starting from %d\n",
        startvertex);
  Graph_BFS (g, startvertex);
  graph_destroy (g);
  return 0;
}
```

papergrid
Date: / /

OUTPUT

Enter the no. of vertices in graph : 4
Enter the no. of edges in the graph: 6
Enter the edges (vertex1 vertex2):

0 1
0 2
1 2
2 0
2 3
3 3

Enter starting vertex for BFS: 2
Following is BFS (starting from 2): 2 0 3 1

## DSF

(2) Write a program to check whether a given graph is connected or not using DSF.

```
#include <stdio.h>
#include <stdlib.h>
# define MAX-EDGE 100
# define MAX_NODE 100
int graph[MAX-NODE][MAX-EDGE];
int visited [MAX_NODE];

void DFS (int start, int n){
    visited[start] = 1;
    for(int i= 0; i<n; i++){
        if (graph [start][i]==1 && !visited[i]){
            DFS(i, n); } } }

int isConnected (int n){
    DFS (0, n);
    for(int i=0; i<n; i++){
```

```
        if (!visited[i]) {
            return 0; }}
    return 1; }

int main() {
    int n, m;
    printf(" Enter the number of nodes & Edges
    scanf("%d %d", &n, &m);
    printf(" Enter the edges:\n");
    for (int i=0; i<m; i++){
        int a, b;
        scanf("%d %d", &a, &b);
        graph[a][b]= 1;
        graph[b][a]= 1; }
    if (isConnected (n)){
        printf(" The graph is connected \n"); }
    else {
        printf(" The graph is not connected \n"); }
    return 0; }
```

OUTPUT

```
Enter the number of edges & nodes : 6 4
Enter the edges:
0 1
0 2
2 3
2 4
4 5
5 1
The graph is connected
```

# Hackerrank : Swap Nodes

```
#include <stdio.h>
#include <assert.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>

typedef struct Node {
    int data,
    struct Node * left,
    struct Node* right; } Node;

Node * createNode (int data){
    Node * newNode = (Node*) malloc (sizeof (Node));
    newNode -> data = data;
    newNode -> left = NULL;
    newNode -> right = NULL;
    return newNode }

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

## OUTPUT

```
3
2   3
-1  -1      ⇒   3 1 2
-1  -1          2 1 3
2
1
1
```

papergrid

Date: / /

Lab - 10

Given a file of N employee records with a set K of Keys (4-digit) which uniquely identify determine the records in file F. Assume that file F is maintained in memory by a Hash table of m memory locations with L as the set of memory addresses (2-digit) of locations in HT.

```
#include <stdio.h>
#include<stdlib.h>
#include <string.h>
# define TABLE_SIZE 100
# define KEY_LENGTH 5
# define MAX_NAME_LENGTH 50
# define MAX_DESIGNATION_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, int m){
    int sum=0;
    for (int i=0; key[i]!='0', i++){
        sum += key[i]; }
    return sum % m; }

void insert (struct Hashtable* ht, struct Employee*
    int index= hash_function(key, TABLE_SIZE)
```

papergrid

Date: / /

```
while (ht -> table[index] != NULL){
    index = (index + 1) % TABLE_SIZE; }
ht -> table[index] = emp; }

struct Employee * search (struct HashTable* ht, const
  char* key) {
int index = hash_function (key, TABLE_SIZE);
while (ht -> table[index] != NULL){
    if (strcmp(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];
    FILE* file;
    char filename[100];
    char line[100];
    for (int i = 0; i < TABLE_SIZE; i++){
        ht.table[i] = NULL; }
    printf ("Enter the filename containing employee
        records");
    scanf ("%s", filename);
    file = fopen(filename, "r");
    if (file == NULL) {
        printf ("Error opening the file");
        return 1; }
while (fgets(line, sizeof(line), file)) {
    emp = (struct Employee *) malloc (sizeof(struct
        Employee));
```

```
sscanf(line, "%s%s%s%f", emp->key, emp->name,
        emp->designation, &emp->salary);
    insert(&ht, emp);
    fclose(file);
    printf("Enter the key to search");
    scanf("%s", key);
    emp = search(&ht, key);

    if (emp != NULL){
        printf("Employee record found with key:
            %s", emp->key);
        printf("Name: %s\n", emp->name);
        printf("Designation: %s\n", emp->designation);
        printf("Salary: %2f\n", emp->salary);
    }
    else {
        printf("Employee record not found for key %s",
            key); }

    for (int i = 0; i<TABLE_SIZE; i++){
        if (ht.table[i] != NULL){
            free(ht.table[i]);
        }
    }
    return 0;
}
```

OUTPUT

```
Enter the filename containing employee records:
    newfile.txt
Enter the key to search: 1
Employee record found with key: 1:
Name: Nidhi
Designation: Developer
Salary: 1250000.00
```

29/2/24