INDEX

Name POOJA.M Std Sec

Roll No. Subject DS observation School/College

Sl. No.	Date	Title	Page No.	Teacher Sign/ Remarks
1	7/12	bob 0 - 8 practice pogms	10	7/12
2	21/12	Lab 1 - 1. pointers swapping 2. Dynamic memory 3. stack implement.	10	21/12
3	28/12/24	Lab 2 - 1. Prefix to postfix 2. Postfix Evaluation 3. queue	10	28/12
4	1/1/24	Lab Week-4 - Singly Linked	10	1/1/24
5	8/1/24	Lab - week-5 - Singly Linked	10	
6	25/1/24	Lab - week 6 - sorting - quick, stalk	10	25/1/24
7	1/2/24	Lab - week 7 - Doubly Link	10	
8	15/2/24	Lab - week 8 - Binary Tree	10	1,2,24
9	22/2/24	Lab - week 9 - BFS	10	
10	29/2/24	Lab week 10 - nothing	10	29/2/24.

10

29/2/24.

papergrid
Date: / /

outputs for practice programs in C

1. Basic Banking System
1. Create Account
2. Withdraw
3. Deposit
4. Check Balance
5. Exit
Enter your choice: 1
Enter your account number: 1904
Enter account holder name: Isabella
Enter initial balance: 700000
Account created successfully!

Basic Banking System.
1. Create Account
2. Withdraw
3. Deposite
4. Check Balance
5. Exit
Enter your choice: 2
Enter the amount of withdrawal: 80000
withdrawal successful. New balance: 620000

Basic Banking System
1. Create Account
2. Withdraw
3. Deposite
4. Check Balance
5. Exit
Enter your choice 5.
Exiting program. Good bye!

2. Enter the number of strings (upto 10):
END
~~Enter 0 strings.~~
OK
BRD
Sorted strings lexicographically
BRD
END
OK.

3. Enter number of rows 2
Enter number of columns 2
Enter the elements of the 2x2 array
3
4
5
6
Enter the element to search for: 7
7 is not present in the array

Date: / /

strings (up to 10):

ographically.

2x2 array:

h for it
ray

papergrid
Date: / /

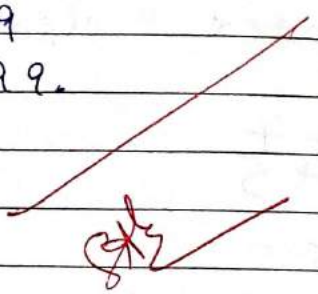
4. Enter the larger string: okay how is life
Enter the substring to search for
: okay
Substring found at index 0 in the larger string

5. Enter the size of the array: 4
Enter 4 elements of the array:
2
44
45
55
Enter the number to find: 44
Last occurence of 44 is at index 1

6. Linear search.
Enter the size of the array: 4
Enter 4 elements of the array:
45
66
55
33
Enter the element to search for: 33
33 found at index 3.

7. binary search.
Enter the size of the sorted array: 4
Enter 4 sorted elements of the array:
23
44
55
66
Enter the elements to search for: 4
4 not found in the array.

8. Enter the size of the array: 5
Enter 5 elements of the array:
67
89
09
88
99
Minimum element : 9
Maximum element : 99.



21/08/23

Swapping 2 numbers using pointers with in a function.

code:-

```
#include <stdio.h>
void swap (int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
int main () {
    int num1, num2;
    printf ("Enter the first number:");
    scanf ("%d", &num1);
    printf ("Enter the second no:");
    scanf ("%d", &num2);
    printf ("Before swapping: num1=%d
        , num2=%d \n", num1, num2);
    swap (&num1, &num2);
    printf ("After swapping: num1=%d,
        num2=%d \n", num1, num2);
    return 0;
}
```

output:-

```
enter the first number 12
enter the second number 33
Before swapping: num1=12, num2=33
After swapping num1=33, num2=12
```

2. Write a program to implement Dynamic memory allocation (like malloc, calloc, free and realloc)

```
#include <stdio.h>
#include <stdlib.h>
void * Malloc (size_t size) {
    return malloc (size);
}
void * Realloc (void *ptr, size_t size)
{ return realloc (ptr, size);
}
void * Calloc (size_t num, size_t size)
{ return calloc (num, size);
}
void Free (void * ptr) {
    free (ptr);
}
int main () {
    int *arr1, *arr2;
    size_t size;
    printf ("Enter the size of the array: ");
    scanf ("%d", &size);
    arr1 = (int*) Malloc (size * sizeof(int));
    if (arr1 == NULL) {
        printf ("Memory allocation failed.\n");
        return 1;
    }
    printf ("Enter elements of the array:\n");
    for (size_t i=0; i<size; i++) {
        printf ("Element %d: ", i+1);
        scanf ("%d", &arr[i]);
    }
```

```
printf ("Elements of the array (malloc):
        \n");
for (size_t i=0; i<size; i++){
    printf ("%d", arr1[i]);
printf ("\n");
size *= 2;
arr2 = (int*) Realloc (arr1, size*
                       size of (int));
if (arr2 == NULL)
    { printf ("Memory reallocation
                failed \n");
      Free (arr1);
      return 1;
    }
printf ("Enter additional elements
                of the array :\n");
for (size_t i = size/2; i<size; i++){
    printf ("Element %d :", i+1);
    scanf ("%d", &arr2[i]);
printf ("\n");
Free (arr2);
return 0;
}
```

Output :-

```
Enter the size of the array : 4
Enter the elements of the
    array :
Element 1 : 3
Element 2 : 4
Element 3 : 5
Element 4 : 6
```

Elements of the array (malloc):
3 4 5 6
Enter additional elements of the array:
Elements 5: 56
Elements 6: 66
Elements 7: 77
Elements 8: 88
Elements of the array (realloc):
3 4 5 6 56 66 77 88

3. Stack implementation [push, pop, display]

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 10

struct Stack {
    int items[MAX_SIZE];
    int top;
};
void initialize(struct Stack* stack) {
    stack->top = -1;
}
int isEmpty(struct Stack* stack) {
    return stack->top == -1;
}
int isFull(struct Stack* stack) {
    return stack->top == MAX_SIZE - 1;
}
void push(struct Stack* stack, int value) {
    if (isFull(stack)) {
        printf("Stack overflow. Cannot push.\n");
```

```
else {
    stack -> top++;
    stack -> items [stack -> top] = value;
    printf ("Pushed %d onto the
                    stack.\n", value);
  }
}
int pop (struct stack *stack) {
    int popped value = -1;
    if (isEmpty (stack)) {
        printf ("stack underflow
               cannot pop from an
               empty stack.\n");
    } else {
    popped Value = stack -> items [stack
                          -> top];
    stack -> top--;
    printf (" Popped %d from the
            stack.\n", poppedValue);
    }
    return popped Value;
}
void display (struct stack *stack)
{   if (isEmpty (stack)) {
        printf ("stack is empty.\n");
    } else {
        printf ("Elements in the
                  stack :");
        for (int i = 0; i <= stack ->
              top ; i++) {
            printf ("%d ", stack -> items[i]);
        }
```

```
    printf("\n");
}
}

int main() {
    struct stack stack;
    initialize(&stack);
    push(&stack, 109);
    push(&stack, 210);
    push(&stack, 31);
    push(&stack, 40);
    display(&stack);
    pop(&stack);
    display(&stack);
    push(&stack, 405);
    display(&stack);
    return 0;
}
```

Output:-

Pushed 109 onto the stack
Pushed 210 onto the stack
Pushed 31 onto the stack
Pushed 40 onto the stack
Elements in the stack : 109 210 31 40
Popped 40 from the stack
Elements in the stack : 109 210 31
Pushed 405 onto the stack
Elements in the stack : 109 210 31 405

29/12/23

1. write a

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 operator, +, -, *, /, ^

code:-

```
#include <stdio.h>
#include <ctype.h>
#define SIZE 50
char stack [SIZE];
int top = -1;
push (char elem)
{
  stack [++top] = elem;
}
char pop()
{
  return (stack [top--]);
}
int pr (char symbol)
{
  if (symbol == '^')
  {
    return (3);
  }
  else if (symbol == '*' || symbol == '/')
  {
    return (2);
  }
  else if (symbol == '+' || symbol == '-')
  {
    return (1);
  }
```

```
else
{
    return 0;
}
}
void main()
{
    char infix[50], postfix[50], ch,
        elem;
    int i=0, k=0;
    printf("enter infix expression");
    scanf("%s", infix);
    push('#');
    while((ch=infix[i++])!='\0')
    {
        if(ch=='(')
        {
            push(ch);
        }
        else
            if(isalnum(ch))
            {
                postfix[k++]=ch;
            }
            else
                if(ch==')')
                {
                    while(stack[top]!='(')
                        postfix[k++]=pop();
                    elem=pop();
                }
                else
                {
```

```
while (pr(stack[top]) >= pr(ch))
    postfix[k++] = pop();
push(ch);
}
}
while(stack[top] != '#')
    postfix[k++] = pop();
postfix[k] = '\0';
printf("Postfix expression = %s", postfix);
}
```

Output :-

enter infix expression (8*5-(12/6^3)+4);

Postfix expression = 85*123^/4+-

2. Postfix Evaluation

code :-

```
#include <stdio.h>
int stack[20];
int top = -1;
void push(int x)
{
    stack[++top] = x;
}
int pop()
{
    return stack[top--];
}
int main()
{
```

```
char exp[20];
char *e;
int n1, n2, n3, num;
printf("Enter expression:");
scanf("%s", exp);
e = exp;
while(*e != '\0')
{
    if(isdigit(*e))
    {
        num = *e - 48;
        push(num);
    }
    else
    {
        n1 = pop();
        n2 = pop();
        switch(*e)
        {
        case '+':
            {
                n3 = n1 + n2;
                break;
            }
        case '-':
            {
                n3 = n2 - n1;
                break;
            }
        case '*':
            {
                n3 = n2 * n1;
                break;
            }
```

3a.

```
case '/':
{
    n3 = n2 / n1;
    break;
}
push(n3);
}
e++;
}
printf("The result of expression = %d", exp, pop());
}
```

Output:-
Enter expression : 7 * 6 ^ 2 +
The result of expression 7 * 6 ^ 2 + = 51

3a. to simulate the working of a queue of integers using an array.

code:-

```
#include <stdio.h>
#define MAX 50
int queue_array[MAX];
int rear = -1;
int front = -1;
display()
{
    int i;
    if (front == -1)
        printf("queue is empty");
    else
```

Date:

```
printf("queue is :\n");
for(i=front; i<=rear; i++)
    printf("%d", queue_array[i]);
printf("\n");
}
}
main()
{
  int choice;
  while(1)
  {
    printf("1. insert");
    printf("2. delete");
    printf("3. display");
    printf("4. exit");
    printf("enter your choice:");
    scanf("%d", &choice);
    switch(choice)
    {
      case 1:
        insert();
        break;
      case 2:
        delete();
        break;
      case 3:
        display();
        break;
      case 4:
        exit();
        break;
      default
      print("invalid choice");
```

```
}
}
insert()
{
    int add_item;
    if (rear == MAX-1)
        printf("queue overflow!");
    else
    {
        if (front == -1)
            front = 0;
        printf("insert the element in
                the queue:")
        scanf("%d", &add_item);
        rear += 1;
        queue_array[rear] = add_item;
    }
}
delete()
{
    if (front == -1 || front > rear)
    {
        printf("queue underflow");
        return;
    }
    else
    {
        printf("deleted element is:
                %d", queue_array[front]);
        front += 1;
    }
}
```

Date: / /

output =
1. insert
2. delete
3. display
4. exit.
enter your choice : 1
insert the element in the queue
1. insert
2. delete
3. display
4. exit
enter your choice : 1
insert the element in the queue
1. insert
2. delete
3. display
4. exit
enter your choice : 3
queue it.
1 2 3
1. insert
2. delete
3. display
4. exit.
enter your choice : 2.
deleted element it : 1 2.
1. insert.
2. delete.
3. display.
4. exit.
enter your choice 3
queue it.
2 3

1. insert
2. delete.
3. display
4. exit.
enter your choice: 4.

3b. Circular Queue Implementation.

code:-

```
#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("Queue full!");
        return;
    }
    q[(++r)%size]=item;
    count++;
}
void dequeue(){
    if (count==0){
        printf("Queue empty!");
        return;
    }
```

Date: / /

```
    rear = (rear+1) % size;
    count++;
}
void display() {
    if (count == 0) {
        printf("Queue empty");
        return;
    }
    int front = f;
    for (int i = 0; i < count; i++) {
        printf("%d ", q[front]);
        front = (front+1) % size;
    }
}
int main() {
    int ch, item;
    while(1) {
        printf("Select choice 1. Enq
                2. Dequeue 3. Disp
        scanf("%d", &ch);
        switch(ch) {
        case 1:
            printf("Enter value to
                    insert");
            scanf("%d", &item);
            enqueue(item);
            break;
        case 2:
            dequeue();
            printf("Item popped");
            break;
        case 3:
            display();
            break;
        default:
```

```
    exit(0);
  }
 }
}
```

output:-

Select choice 1. Enqueue 2. Dequeue
3. Display 1

Enter value to insert : 12

Select choice 1. Enqueue 2. Dequeue
3. Display 1

Enter value to insert : 23

Select choice 1. Enqueue 2. Dequeue
3. Display 1
Enter value to insert : 34.
Select choice 1. Enqueue 2. Dequeue
3. Display 2.
Item popped
Select choice 1. Enqueue 2. Dequeue
3. Display 2
Item popped.
Select choice 1. Enqueue 2. Dequeue
3. Display 3.
23
Select choice 1. Enqueue 2. Dequeue
3. Display 2.
Item popped.
Select choice 1. Enqueue 2. Dequeue
3. Display 2.
Queue empty!

Date: / /

WEEK - 4

Singly Linked List Insert and display Implementation.

code:-

```
#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");
        return;
    }
    printf("Elements 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
    printf("Enter the value to be inser
    scanf("%d",&temp->data);
    temp->next=head;
    head=temp;
}
void insert_end(){
    struct node *temp,*ptr;
```

display

```
temp = (struct node *) malloc (sizeof
                    (struct node));
printf ("Enter the value to be
        inserted :");
scanf ("%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");
    return;
}
temp->next = ptr->next;
ptr->next = temp;
}

int main() {
    int choice;
    while(1) {
        printf("1. Insert at the beginning
                2. Insert at the end
                3. Insert at any position
                4. Display 5. Exit");
        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
                    \n");
    }
}
return 0;
}
```

output:-

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

Enter your choice: 1
Enter the value to be enserted: 12.

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

Enter your choice: 3
Enter your value to be inserted: 56.

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

Enter your choice: 4.
Elements are: 12 56.

Date: / /

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

1. Insert at the beginning
2. Insert at the end
3. Insert at any position
4. Display
5. Exit
Enter your choice :
5

8/11/24

Lect wde [11-04-24]

```
#include <stdlib.h>
typedef struct {
int *stack;
int *minstack;
```

```
int top;
} MinStack;

MinStack* minStackCreate() {
MinStack * stack = (MinStack *) malloc
                    (sizeof(MinStack));
stack -> stack = (int*) malloc (sizeof(int)
                    * 50);
stack -> minStack = (int *) malloc (sizeof(int
                    ) * 50);
stack -> top = -1;
return stack;
}

void minStackPush (MinStack * obj, int val) {
obj -> top ++
obj -> stack [obj -> top] = val; }
else {
obj -> minStack [obj -> top] = obj -> minStack
                    [obj -> top -1];
}
}
void minStackPop (MinStack * obj) {
obj -> top --;
}
int minStackTop (MinStack * obj) {
return obj -> stack [obj -> top];
}
int minStackGetMin (MinStack * obj)
{
return obj -> minStack [obj -> top];
}
void minStackFree (MinStack * obj) {
```

```
free(obj->stack);
free(obj->minStack);
free(obj);
}
```

output:

Input

["MinStack", "push", "push", "push", "getMin", "pop", "top", "getMin"]

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

Output

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

Expected

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

Week 5

Singly Linked List Delete and display

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    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_begin() {
    if (head == NULL) {
        printf("List is empty. Deletion
            not possible!\n");
        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. Deletion is not possible.");
        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. Deletion
            not possible");
    return;
}
struct node *temp, *prev;
temp = head;
if (position == 0) {
    head = head -> next;
    printf("Element at position %d
            deleted successfully",
            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
            bound.\n", position);
    return;
}
prev -> next = temp -> next;
printf("Elements at position %d
        deleted successfully", position);
free(temp);
}
int main() {
    int choice;
    while (1) {
```

```
printf("1. to insert at the beginning
        2. to delete beginning
        3. to delete in end
        4. to delete at a position
        5. to display 6. to exit");
printf("Enter your choice");
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 the correct choice");
        break;
}
}
while(1);
}
```

outputs.

1. to insert at the beginning
2. to delete beginning
3. to delete at end
4. to delete at any position.
5. to display
6. to exit.

Enter your choice: 1
Enter the value to be inserted: 12.

1.
2.
3.
4.
5.
6.

Enter your choice: 1
Enter the value to be inserted: 21

1.
2.
3.
4.
5.
6.

Enter your choice: 1
Enter your the value to be inserted: 32.

1.
2.
3.
4.
5.
6.

IRN No: .....

paperg

Date: / /

Enter your choice : 5
Elements are : 32 → 21 → 12 → NULL

1. to insert at beginning.
2. to delete beginning.
3. to delete at end
4. to delte at position.
5. to display
6. to exit.

Enter your choice : 2.
Element deleted from the begging.

1. to insert at the beginning
2. to delete beggining
3. to delete at end
4. to delete at any position
5. to display.
6. to exit.

Enter your choice : 3
Element deleted from thend : 12

1.
2.
3.
4.
5. to display
6. to exit.

Enter your choice : 5
Elements are : 67 → 45 → NULL

1. to insert
2. to delete from beggining
3. to delete from end

4. to dclt at any position
5. to display
6. to exit
Enter your choice : 3
Element deleted from the end : 45.

1.
2.
3.
4.
5.
6.
Enter your choise : 5
Elements are : 23 → 67 → NULL

1.
2.
3.
4.
5.
6. to exit
Enter your choice : 6.

Lect code - 2.

```
struct ListNode* reverseBetween
    (struct ListNode* start, int a
     int b)
{
    a -= 1;
    b -= 1;
    struct ListNode *node1=NULL,
    *node2=NULL, *nodeb=NULL,
    *nodea=NULL, *ptr=start;
```

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

while (ptr!=NULL)
{
    if (c>=a && c<b)
    {
        temp=ptr->next;
        ptr->next=prev;
        prev=ptr;
        ptr=temp;
    }
    else if (c==b)
    {
        ptr->next=prev;
```

```
if (a==0)
    start=ptr;
else
    node->next=ptr;
    ptr=ptr;
}
else
    ptr=ptr->next;
    i++;
}
return start;
}
```

Output:
head
[1,2,3,4,5]
left =
2
right =
4
Output
[1,4,3,2,5]
Expected
[1,4,3,2,5].

WEEK - 6

Singly linked list with following operations
a) reverse, b) concatenation.

```
write:-
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node *next;
};
struct Node *createNode(int val)
{
    struct Node *newNode = (struct
        Node *)malloc(sizeof(
        Node));
    newNode->data = value;
    newNode->next = NULL;
    return newNode;
}
void insertEnd(struct Node **head,
               int value) {
    struct Node *newNode = createNode
        Node(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");
}
void sortLinkedList (struct Node* head) {
    int swapped, i;
    struct Node *ptr;
    struct Node *ptr = NULL;
    if (head == NULL)
        return;
    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);
}
struct Node* reverseLinkedList
    (struct Node* head)
```

```
{
struct Node *prev = NULL, *
        current = head, *next;
while (current != NULL) {
next = current -> next;
current -> next = prev;
prev = current;
current = next;
}
return prev;
}
void concatenate LinkedList (struct
        Node **list1, struct Node *li
if (*list1 == NULL) {
*list1 = list2;
} else {
struct Node *temp = *list1;
while (temp -> next != NULL)
        temp = temp -> next;
}
temp->next = list2;
}
}
int main() {
struct Node *list1 = NULL;
struct Node *list2 = NULL;
int n, value;
printf ("Enter the number of
        elements for list1:");
scanf ("%d", &n);
printf ("Enter the elements for
        list1:\n");
```

```
for (int i=0; i<n; i++){
    scanf("%d", &value);
    insertEnd(&list1, value);
}
printf("Enter the number of elements
        for the list 2:\n");
scanf("%d", &n);
printf("Enter the elements for
        list 2:\n");
for(int i=0; i<n; i++){
    scanf("%d", &value);
    insertEnd(&list2, value);
}
sortLinkedList(list1);
printf("Sorted List 1:\n");
display(list1);
list2 = reverseLinkedList(list2);
printf("Reversed List 2:\n");
display(list2);
concatenateLinkedList(&list1, list2);
printf("Concatenated List:\n");
display(list1);
struct Node* temp;
while(list1 != NULL){
    temp = list1;
    list1 = list1->next;
    free(temp);
}
return 0;
}
```

Output:-

Enter the number of elements for list1: 3

Enter the elements for list 1:
12
23
45.
Enter the number of elements for list 2: 3.
Enter the elements for list 2:
33
26
67
Sorted List 1: 12 → 23 → 45 → NULL
Reversed List 2: 67 → 26 → 33 → NULL
Concatenated List: 12 → 23 → 45 → 67 → 26 → 33 → NULL.

Singly LinkList with stack.

code:-

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
struct Node* createNode(int value
{ struct Node* newNode = (stru
    node(*)malloc(sizeof(struct Nod
newNode->data = value;
newNode->next = NULL;
return newNode;
}
int pop(struct Node** top) {
```

```
if (*top == NULL) {
    printf ("Stack underflow!");
    return -1;
}
struct node* temp = *top;
int poppedvalue = temp->data;
*top = temp->next;
free (temp);
return poppedvalue;
}
void display stack (struct node* top) {
    printf ("stack");
    while (top != NULL) {
        printf ("%d", top->data);
        top = top->next;
    }
    printf ("\n");
}
int main () {
    struct node* top = NULL;
    int choice, value;
    do {
        printf ("Stack operation");
        printf ("1. Push");
        printf ("2. pop");
        printf ("3. Display");
        printf ("4. Exit");
        printf ("Enter your choice");
        scanf ("%d", &choice);
        switch (choice) {
        case 1:
            printf ("Enter the value
                to push");
```

```
scanf("%d", &value);
push(&top, value);
break;
case 2:
value = pop(&top);
if (value != -1)
    printf("Popped value = %d\n", value);
}
break;
case 3:
displayStack(top);
break;
case 4:
printf("Exiting the program\n");
break
default:
printf("Invalid choice!
        please a valid option\n");
}
} while (choice != 4);
struct Node *temp;
while (top != NULL) {
    temp = top;
    top = top->next;
    free(temp);
}
return 0;
}
```

Output :-

Stack operations

1. Push

2. pop
3. Display
4. Exit
Enter your choice: 2
Enter value to push: 2.
Stack operations:
1. push.
2. pop.
3. Display
4. Exit
Enter your choice: 1
Enter the value to push: 3
Stack operation
1
2
3
4
Enter your choice: 1
Enter the value to push: 4
Stack operations.
1.
2.
3.
4.
Enter your choice: 2.
popped value: 4.

Stack operations.
1.
2.
3.
4.
Enter your choice: 3
Stack: 32

Singley linked implementation of Queue

code:-

```
#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
    malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;
    return newNode;
};
struct queue* createQueue() {
    struct Queue* queue = (struct Que
    malloc(sizeof(struct Queue));
    queue->front = queue->rear = NULL;
    return queue;
};
void enqueue(struct Queue* que
             int value) {
    struct Node* newNode = create
                           Node(value);
    if(que->rear == NULL) {
        que->front = que->rear
                   = newNode;
        return;
};
```

```
queue->rear->next = newNode;
queue->rear = newNode;
}

int dequeue(struct Queue* queue){
    if (queue->front == NULL){
        printf("Queue overflow");
        return -1;
    }
    struct Node* temp = queue->front;
    int dequeuedValue = temp->data;
    queue->front = temp->next;
    if (queue->front == NULL){
        queue->rear = NULL;
    }
    free(temp);
    return dequeuedValue;
}
void displayQueue(struct Queue* queue){
    struct Node* temp = queue->front;
    printf("Queue: ");
    while (temp != NULL){
        printf("%d", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
int main(){
    struct Queue* queue = (struct Queue*)
    int choice, value;
    do {
        printf("Queue operations");
        printf("1. Enqueue");
        printf("2. Dequeue");
```

```
printf("4. Exit");
printf("Enter your choice:");
scanf("%d", &choice);
switch(choice) {
case 1:
printf("Enter the value to
        enqueue:");
scanf("%d", &value);
enqueue(que, value);
break;
case 2:
value = dequeue(que);
if(value != -1) {
    printf("Dequeued value", va
}
break;
case 3:
display queue(que);
break;
case 4:
printf("Exiting the prgm");
break;
default:
printf("Invalid choice, Pleas
        enter a valid option");
}
} while(choice != 4);
struct Node* temp;
while(que->front != NULL) {
temp = que->front;
que->front = que->front
        ->next;
free(temp);
}
```

```
free(queue);
return 0;
}
```

output:-

Queue operations.
1. Enqueue
2. dequeue.
3. Display
4. Exit.
Enter your choice: 1.
Enter your value for enqueue: 2

repeat for value 3, 4, 5 by choosing option 1.

Queue operations
1. Enqueue
2. dequeue
3. Display
4. Exit
Enter your choice 2.
Dequeue value: 2

Queue operations.
1. Enqueue
2. dequeue
3. Display
4. Exit
Enter your choice: 3
Queue: 3 4.

Queue operations.
1. Enqueue.
2. Dequeue
3. Display.
4. Exit
enter your choice: 4.
Exiting the program.

1/2/24.

WEEK-7:

Doubly link list (creation, insertion, deletion, display:

```
code:-
#include<stdio.h>
#include<stdlib.h>
struct node{
    int data;
    struct node *prev;
    struct node *next;
};
struct node *s1 = NULL;
struct node* createNode(int value)
    struct node *temp = (struct node*)
                malloc(sizeof(struct
    temp->data = value;
    temp->next = NULL;
    temp->prev = NULL;
    return temp;
}
struct node *insertLeft(struct node*
                start){
    int value, key;
    struct node *temp = createNode(va
    printf("Enter the value to be inserted
    scanf("%d", &temp->data);
    printf("Enter the value to the le
            of which the node has to be inse
    scanf("%d", &key);
    struct node *ptr = start;
    while(ptr != NULL && ptr->da
        ptr = ptr->next;
}
```

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

return start;

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

```
        ptr->next->prev = ptr->prev;
    }
    printf("Node with value %d deleted", ptr->data);
    free(ptr);
}
return start;
}
void display(struct node* start){
    struct node *ptr = start;
    if(start == NULL)
        printf("List is empty\n");
    } else {
        printf("List contents: \n");
        while(ptr != NULL){
            printf("%d ", 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;
("\n",value);
}
return start;
}
int main()
{
int choice;
while(1)
{
printf("\n1. Create a doubly linklist
\n2. Insert to the left of a node
\n3. Delete based on a specific
value \n4. Display the
contents \n5. Insert to
right \n6. Exit");
scanf("%d",&choice);
switch(choice)
{
case 1:
s1=createNode(0);
printf("\nDoubly Linked 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 the program");
exit(0);
```

default :
printf ("Invalid choice");
}
return 0;
}

Output :-
1. Create a doubly linked list.
2. Insert to the left of a node
3. Delete based on a specific value
4. Display based contents.
5. Insert to right.
6. Exit.
Doubly linked list created.

1. Create doubly linked list.
2. Insert to left of a node.
3. Delete based on a specific
4. Display contents.
5. Insert to right.
6. Exit 5.
Enter the value to be insert 19

1.
2.
3.
4. Display contents.
5.
6.
List contents :
0
16
18
19

3. Delete based on a specific

Enter the node with value to deleted.

3.

1. Display the contents.

ist contents:

0.

16.

9.

Create a doubly

. Insert to the left.

. Delete based on a specific value.

. Display the contents.

5. Insert to right.

. Exit 6.

xiting the program.

ect code :-

include <stdlib.h>

rut List Node ** split list Topark (

struct List Node * head, int K,

int * return Size) {

struct List Node * current = head;

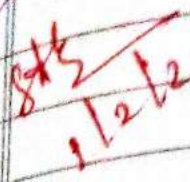
```
ListNode
    int length = 0;
    while (current) {
        length ++;
        current = current -> next;
    }
    int part_size = length / k;
    int extra_nodes = length % k;
    struct ListNode **result = (struct
    ListNode **) malloc (k * sizeof(
    ListNode *));
    current = head;
    for (int i = 0; i < k; i++) {
        struct ListNode *part_head =
            current;
    int part_length = part_size + (i < extra_n
    for (int j = 0; j < part_length - 1
        current = current -> next;
    }
    if (current) {
        struct ListNode* next_node =
            current -> next;
        current -> next = NULL;
        result[i] = part_node;
    } else {
        result[i] = NULL;
    }
}
*returnSize = k;
return result;
}
```

Output:

input [1, 2, 3]

output:
[[1], [2], [3], [3, [7]]

Expected
[[1], [2], [3], [7, [7]]



WEEK-8.

papergrid
Date: / /

Binary Tree

```
#include <stdio.h>
#include <stdlib.h>
  struct TreeNode {
      int data;
      struct TreeNode *left;
      struct TreeNode *right;
  };
struct TreeNode *createNode(int
      data) {
    struct TreeNode *newNode = (struct
        TreeNode)*malloc(sizeof(struct Tr
    newNode->data=data;
    newNode->left=newNode->right
    return newNode; }
struct TreeNode *insertNode(struct Tree N
    if (root==NULL) {
        return createNode(data); }
    if (data<root->data) {
        root->left=insertNode(root->left
    else if (data>root->data) {
        root->right=insertNode(root->
    return root; }
void inOrderTraversal(struct TreeNode *root
    if (root!=NULL) {
        inOrderTraversal(root->left);
        printf("%d", root->data);
        inOrderTranversal(root->right);
void preOrderTranversal(struct TreeNode *
    if (root!=NULL) { postOrderTrave
        postorderTraversal(root->right);
        printf("%d", root->data); } }
void postOrderTraversal(struct TreeNode *ro
    if (root!=NULL) { postOrderTrave
        postorderTranversal(root->right)
        printf("%d", root->data); } }
```

```
void displayTree(struct TreeNode *root){
    printf("In order Traversal:");
    inOrderTraversal(root); printf("\n");
    printf("Pre-order Traversal");
    preOrderTraversal(root); printf("\n");
    printf("Post-order traversal");
    postOrderTraversal(root); }
int main(){
    struct TreeNode *root = NULL;
    int choice, data;
    do{
        printf("1. Insert a node");
        printf("2. Display");
        printf("3. Exit");
        printf("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");
            }else{
                displayTree(root); }
            break;
        case 3:
            printf("Exiting program");
            break;
        default printf("Invaild choice"); }
    } while(choice != 3);
    return 0; }
```

output:-

repeat if 2 more time

1. Insert a node.
2. Display treee
3. Exit.

Enter your choice: 1
Enter your data to insert: 12

Enter your data to insert: 34.
Enter your data to insert: 56.
Enter your choice: 2.
In-order traversal: 12 34 56.
Pre-order traversal: 12 34 56
Post order traversal: 56 34 12.

Exit.

LEET CODE.

```
struct ListNode* rotateRight(struct Li
    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;
    for (int i = 1; i < length - k; i ++){
        current = current -> next; }
    struct ListNode* newHead = current;
```

```
next;
current = newHead;
while (current -> next != NULL) {
    current = current -> next; }
current -> next = head;
return newHead; }
```

Output :-

head =

[1, 2, 3, 4, 5]

k =

2

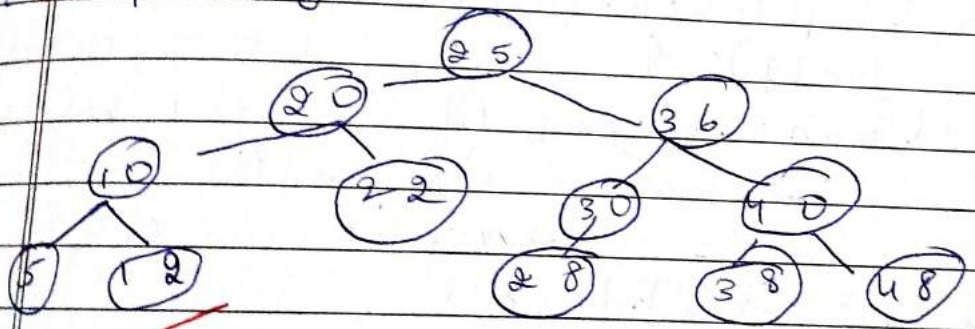
output:

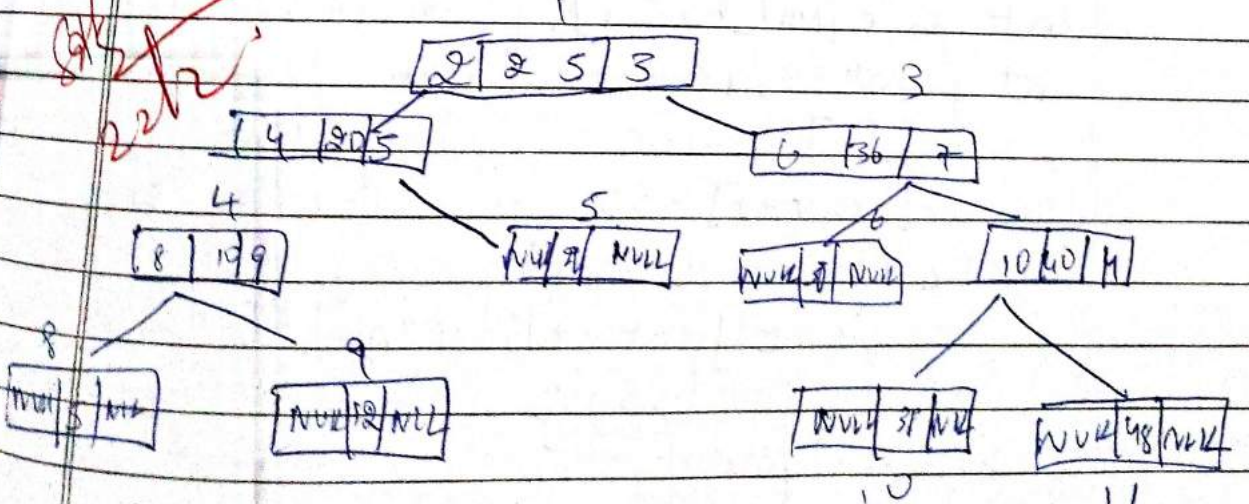
[4, 5, 1, 2, 3]

Expected

[4, 5, 1, 2, 3].

Tracing





To traverse a graph using BFS method.

Date: / /

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#define MAX_VERTICES 50
typedef struct Graph_t {
int V;
bool adj[MAX_vertices][MAX_Vertices];
} 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_BFS(Graph* g, int s)
{bool visited[MAX_VERTICES];
  for (int i = 0; i < g->V; i++)
{ visited[i] = false; }
int queue[MAX_Vertices];
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;
}
}
void Graph_addEdge(Graph* g,
int v, int w) {
g->adj[v][w] = true;
}
int main() {
int numVertices;
printf("Enter the number
of vertices in the graph");
scanf("%d", &numVer
Graph* g = Graph_cre
(numVertices);
int numEdges;
printf("Enter the numbe
of edges in the graph
scanf("%d", &numEdg
printf("Enter the edges
(vertex1 vertex2)
for (int i = 0; i < numEd
int v, w;
scanf("%d %d", &v,
Graph_addEdge(g, v,
int startVertex;
printf("Enter the starti
vertex for BFS: ");
scanf("%d", &start
printf("Following is
Breadth First Trave
startVertex);
Graph_BFS(g,
Graph_destroy(g);
return 0;
}
```

Output:

```
Enter no of vertices in the 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 the starting vertex for BFS
Following is Breadth first Tranversal
2 0 3 1
```

– To check if it connected or not.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_NODES 1001
#define MAX_EDGES 100
int graph[MAX_NODES][MAX_NODES];
int visited[MAX_NODES];
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 and edges:");
    scanf("%d %d",&n,&m);
    printf("Enter the edges");
    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");
    }else{
        printf("The graph is Not connected");}
    return 0;}
```

Output:

```
Enter the no of vertices : 5
Enter the no of edges : 4
Enter edges (src dest):
0 1
1 2
2 3
3 4
The graph is connected.
```

Hackerrank.

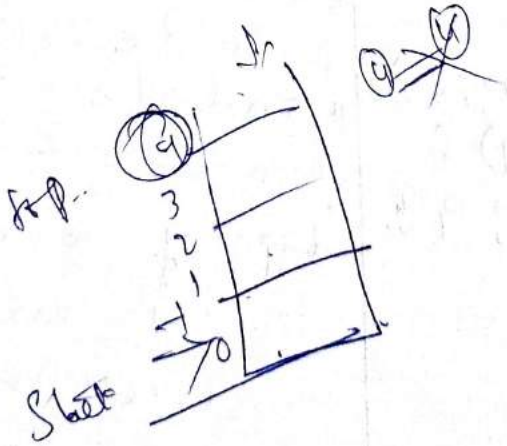
```
#include<assert.h>
#include<stdbool.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
typedef struct Node{
    int data;
    struct Node* left;
    struct Node* right;
} Node;
Node* createNode(int data) {
    Node* newNode
        = (Node*)
        malloc(sizeof(Node));
```

```
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);
}
void swapAtLevel(Node* root,int k,int level){
    if(root==NULL)
    if(level%k==0){
        Node* temp=root->left;
        root->left=root->right;
        root->right=temp;
    }
    swapatlevel(root->left,k,level+1);
    swapatlevel(root->right,k,level+1);
```

Output:

3 1 2

2 1 3



Hashing

```
#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* emp) {
    int index = hash_function(emp->key
            , TABLE_SIZE);
    while (ht->table[index] != NULL) {
        index = (index + 1) % TABLE_SIZE;
    }
    ht->table[index] = emp;
}
```

```
        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 file");
        return 1;
    }
    while (fgets(line, sizeof(line), file)
        emp = (struct Employee*) malloc
                (sizeof(struct Employee
        sscanf(line, "%s %s %s", emp
        emp->name, emp->designation);
        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 %d", emp->key);
    printf("Name : %s", emp->name);
    printf("Designation : %s\n", emp->
        designation);
    printf("Salary : %.2f\n", emp->
        salary);
} else {
    printf("Employee record not
        found for key %d", 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 : textfile.txt

Enter the key to search : 2

Employee record found with Key 2.

Name : pooja

Designation : analyst

Salary : 400000.00

29/2/24