INDEX

Name: Tooja. Gaikwad

Roll No. Subject School/College

School/College Tel No Parents Tel No

Sl. No.	Date	Title	Page	Sign
1	7/12/23	LAB-1 pratice program	10	
2	21/12/23	LAB-2 swapping, Realloc, malloc & push, pop functions.	10	
3	28/12/23	LAB-3 1] Infix - to post fix 2] post fix evalution	10	
		LAB-4 3] queue.		
4	11/01/24	circular queue & linked single list	10	
5	18/01/24	LAB-5 1] singly linked list delete & display implementation and leet code prgm.	10	18/1/24
6.	25/01/24	LAB-6 1] sort, reverse concatation of lists 2] stack 3] queue.	10	
7.	01/02/24	* Double link list.	10	1/2/24
8.	15/02/24	Binary search tree.	10	22/2/24
9	22/02/24	BSF & DSF	10	
10.	29/02/24	Hashing	10	29/2/24

10

29/2/24

LAB - 1 :

7/21

1. Develop a c program that simulates a basic banking system with functionalities like account creation, withdrawal, deposit, and balance inquiry write different user-defined function for each

→ Output :-

*** Basic Banking system menu ***

1. Create Account
2. Withdraw
3. Deposit
4. Balance Inquiry
0. Exit

Enter your choice : 1

Enter Account number : 12345678

Enter Account holder name : Pooja Gaikwad

Enter Initial Balance : 475000

2. Implement a C program that sorts strings lexicographically considering uppercase & lowercase letters, & without using the standard library sorting functions

→ Enter the number of strings (up to 10) : 4

Enter 4 strings :

Pooja

tannu

nidhi

ritika

sorted strings :

nidhi

Pooja

ritika

tannu

3. Implement a C program to check if a given element is present in a 2D array with a user defined function

→ Output: Enter the number of rows & columns (up to 10 each)
Enter the elements of the 2D array:
Element at position [0][0]: 1
Element at position [0][1]: 2
Element at position [0][2]: 3
Element at position [1][0]: 4
Element at position [1][1]: 5
Element at position [1][2]: 6
Enter the element to search: 5
Element 5 is present in the 2D array.

4. Create a program in C to search for substring within a larger string with a user defined function.

→ Output: Enter the larger string: Pooja gaikwad
Enter the substring to search for: gai
substring found at index 6

5. Write a C program to find the index of the last occurrence of a number in an array with a user defined function

→ output: Enter the size of the array: 5
Enter the array elements:
1 2 3 2 4
Enter the number to find: 2
Last occurrence of 2 is at index 3

6. write a c program to search for a specific element in an array using linear search with a user defined function.

→ output: Enter the size of the array : 5
Enter 5 elements :
10 20 30 40 50
Enter the element to search : 30
Element 30 found at index 2.

7. Implement a c program to perform a binary search on a sorted array with a user defined function.

→ output: Enter the size of the array : 5
Enter the elements of the array in sorted order:
1 2 3 4 5
Enter the element to be searched : 3
Element 3 found at index 2.

8. Create a program in c to search for the minimum and maximum elements in an array with a user defined function.

→ output: Enter the size of the array : 5
Enter 5 elements :
Element 1: 1 2 3 4 5
Minimum element : 1
Maximum element : 5

LAB - 08 :-

Q. Write a program for the swapping two programs using points with a function

```
# 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 \n");
 scanf ("%d", & num1);
 printf (" Enter the second number \n");
 scanf ("%d", & num2);
 printf (" Before swapping : num1 = %d
        num2 = %d \n", num1, num2);

swap (& num1, & num2);
printf (" After swapping : num1 = %d, num=2
        %d \n", num1, num2);

return (0);
}
```

=> Output:

Enter the first number = 10
enter the second number = 25
before swapping num1 = 10, num2 = 25
after swapping num1 = 25, num2 = 10

8. Write a program to implement Dynamic Memory Allocation use malloc, calloc, free and Realloc

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

void * myRealloc (size_t size) {
        return malloc (size);
}
void * myRealloc (void * ptr, size_t size)
{
    return realloc (ptr, size);
}
void* My calloc (size_t num, size_t size) {
    return calloc (num, size);
}
void my free (void * ptr) {
        free (ptr);
}
int main () {
        int *arr1, *arr2;
        size_t size;
        printf (" Enter the size of the array :");
        scanf ("%zu", &size);

    arr1 = (int*) my 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 %zu: ", i + 1);
    scanf ("%d", &arr1[i]);
}
printf ("Elements of the array (malloc) : \n");
for (size_t i = 0; i < size; i++) {
    print ("%d", arr2[i]);
}
printf ("\n");

size *= 2;
arr2 = (int*) myRealloc (arr1, size * sizeof
    (int));

if (arr2 == NULL)
{
printf ("Memory reallocation failed. \n");
    my free (arr2);
    return 1;
}
printf
printf ("Enter additional elements of the array:
\n");

For (size_t i = size/2; i < size; i++) {
    scanf ("%d", &arr2[i]);
}
printf ("Elements of the array (realloc): \n");
for (size_t i = 0; i < size; i++)
{
    printf ("%d", arr2[i]);
}
printf ("\n");
```

```
    My_free(arr2);
    return 0;
}
```

→ Output:

```
Enter the size of the array : 5
Enter elements of the array:
 Element 1: 1
 Element 2: 2
 Element 3: 3
 Element 4: 4
 Element 5: 5
Elements of the array (malloc):
  1 2 3 4 5
Enter aditional elements of the array:
 Element 6: 11
 Element 7: 22
 Element 8: 33
 Element 9: 44
 Element 10 : 55
Elements of the array (realloc):
  1 2 3 4 5 11 22 33 44 55
```

Q.3. write a c program for stack implementation which push, POP, display functions to be implemented.

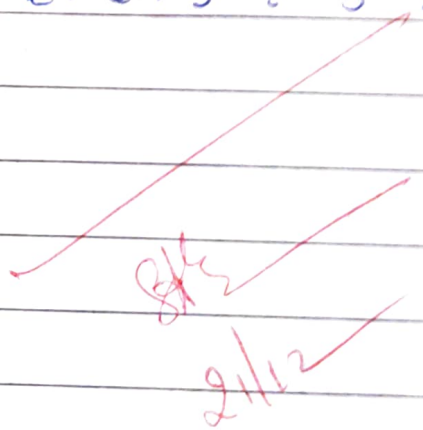
→
```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 10
struct stack {
    int arr[MAX_SIZE];
    int top;
};
```

```
void initialize stack (Struct stack * stack) {
        stack -> top = -1;
    }
int isEmpty (struct stack * stack)
    {
     return stack -> top == -1;
    }
int is full (Struct stack * stack) {
     return stack -> top == MAX_SIZE - 1;
}
void push (Struct stack *stack, int value) {
    if (is Full (stack)) {
      printf (" stack overflow - cannot push elem
%d \n", value);
      return ;
    }
    stack -> arr [++ stack -> top] = value;
    printf (" pushed %d onto the stack. \n",
            value);
}
void pop (Struct stack * stack) {
      if (is Empty (stack)) {
      printf (" stack underflow: cannot pop from
an empty stack. \n");
      return ;
    }
    printf (" popped %d from the stack. \n", stack ->
    arr [stack -> top--]);
}
void display (Struct stack * stack)
```

```
{
if (is empty (stack)) {
    printf ("stack is empty. \n");
    return;
}
printf ("stack elements :");
for (int i = 0; i <= stack -> top; i++)
{
  printf ("%d ", stack -> arr [i]);
}
 printf ("\n");
}
int main () {
    struct stack mystack;
    initialize stack (& mystack);

    push (& my stack, 5);
    push (& my stack, 10);
    push (& my stack, 15);
    display (& my stack);

    Pop (& my stack);
    display (& my stack);

    Push (& Mystack, 20);
    display (& my stack);

    return 0;
}
```

→ output :

pushed 5 onto the stack
pushed 10 onto the stack
pushed 15 onto the stack
stack elements : 5 10 15
stack elements : 5 10
pushed 20 onto the stack
stack elements : 5 10 20



LaB - 03 :—

28/11/23

1. Write a pgm to convert a given valid pareth -esized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus) - (minus), * (multiply), / (divide) and ^ (power).

→

```
# 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 (1);
    }
    else
```

```
    {
      return(1);
    }
    else
    {
      return (0);
    }
}
void main()
  {
    char infix[50], postfix[50,], ch, elem;
    int i=0, k=0;
    printf("Enter the 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("\n Postfix Expression = %s\n",
                    postfix);
}
```

→ output :

Enter infix expression (8*5-(12/6^3)*4)^3

postfix expression = 85*1263^/4*-3^

Q2: write a program for postfix Evalut -ion.

→

```
#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 the 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 = n1 + n2;
      break;
}
```

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

→ output:-

Enter expression : 12*34*+5-

the result of expression 12*34*+5- = 9

3] Q. WAP to simulate the working of a queue of integers using an array. provide the following operations: Insert, delete, display. The program should print appropriate messages for queue empty and queue overflow conditions.

→
```
#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 \n");
else
  {
   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 \n");
  printf ("2. delete \n");
  printf ("3. display \n");
```

```
printf("\n\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch(choice)
{
    case 1:
    insert();
    break;
    case 2:
    display();
    break;
    case 3:
    display();
    break;
    case 4:
    exit(0);
    break;
    default:
    printf("Invalid choice\n");
}
}
insert()
{
    int num;
    if(rear==MAX-1)
    printf("Queue overflow\n");
    else
    {
    if(rear==-1)
    front=0;
```

```
printf("insert the element in the queue :");
scanf("%d", &add_item);
rear++;
queue_array[rear] = add_item;
}
}

delete()
{
if (front == -1 || front > rear)
{
printf("queue underflow\n");
return;
}
else
{
printf("deleted element is : %d\n",
queue_array[front]);
front += 1;
}
}
```

→ output :

1. insert
2. delete
3. display
4. exit

enter your choice : 2
inser the element in the queue : 15
1. insert
2. delete
3. display
4. exit

enter your choice : 2
deleted element is - 15

3.b. circular queue :—

```
#include <stdio.h>
#define SIZE 5
int items [SIZE];
int front = -1, rear = -1,
int isFull()
if ((front == rear+1) || (front == 44 rear ==
        size -1)
return 1;
return 0;
}
int isEmpty !
    if (front == -1)
        return 1;
        return 0;
}
void enqueue (int element) {
if (is true)
printf (" In queue is free");
else {
    if (front == -)
    front = 0;
```

```
items = [rear] = element
printf("\n Inserted = %d", element);
}
}

int dequeue() {
    int element
    if (isEmpty()) {
printf("\n Queue is empty");
    return element;
    }
}

void display() {
    int i;
    if (isEmpty())
printf("Empty queue();
    else {
    printf("front → %d", front);
    printf("items -");
    for (i = front; i = rear; i = (i+1) % SI
    {
    printf("%d", items[i]);
    printf("Rear → %d\n", rear);
    }
}

int main() {
    enqueue(1);
    enqueue(2);
    enqueue(3);
    enqueue(4);
    enqueue(5);
    display();
    dequeue();
    display()
```

```
enqueue(6);
enqueue(7);
display();
return 0;
}
```

→ output :

```
Inserted 1
Inserted 2
Inserted 3
Inserted 4
Inserted 5
Front - 0
Items - 1, 2, 3, 4 5
Rear 4

Deleted element 1
Deleted element 2.
```

Monday LAB 4

Singly linked list :-

```
#include <stdio.h>
#include <malloc.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 (size of (struct
    node));
    printf ("Enter value to be inserted");
    scanf ("%d", &temp -> data);
    temp -> next = head;
    head = temp;
void insert_end () {
    struct node * temp1 * ptr;
```

```
temp = (struct node *) malloc (size of (struct node));

printf ("Enter value");
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 (size of (struct
                        node));
printf ("Enter position");
scanf ("%d", &pos);
printf (" Enter value");
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;
```

```
default :
    printf ("Enter correct choice");
}
}
return 0;
}
```

→ output :

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

Enter your choice = 1
Enter the value to be inserted : 9

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

Enter your choice = 1
Enter value : 5

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

Enter your choice = 2
Enter value : 6

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

Enter your choice : 4
Elements are 5 9 6

1. Inser at beginning
2. inert at end
3. Insert at position
4. Display
5. exit

Enter your choice : 3
Enter position : 2
Enter value : 8

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

Enter choice : 4
Elements are 5 9 8 6

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

Enter your choice : 5

```
#include <stdio.h>
typedef struct {
int * stack;
int * Min stack;
int top
} minstack;

Minstack * minstack Create () {
Minstack * stack = (minstack*) malloc (size of
  Minstack));
stack → stack = (int*) malloc (size of int) *
  10,000);
stack → minstack = (int*) malloc (size of (int) *
  10,000);
stack → top = -1;
  return stack;
}
void Minstack push (minstack * obj, int val) {
  obj => top ++;
obj → stack [obj → top] = val;
if [obj → top = oll val <= obj → minstack
    [obj → top] = val;
}
else
{
obj → minstack [obj → top] = obj → minstack
  [obj → top - 1];
}
}
void minstack pop (minstack * obj) {
  obj → top --;
```

```
}
int minstack top (minstack * obj;)
{
return obj; → stack [obj; - top ]
}
int minstack Get Min (minstack *obj;]
  {
  return obj; → Min stack [obj; → top] ;
}
void minstack free (Minstack * obj ]
{
free (obj; → stack);
free (obj; → minstack);
free (obj;);
}
```

output :

- case 1.

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]

18/12

18/11/24

LAB - 5 :-

* WAP To Implement singly linked list with follo -wing operations

a) create a linked list
b) Deletion of first element, specified element and last element in the list.
c) Display the contents of the linked list

→
```
# include < stdio.h >
# include <stdlib.h>
struct node {
    int data ;
    struct node * next ;
};
struct node * head = NULL ;
void display () {
    printf ("Element 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 intersed :") ;
    scanf ("%d", & temp → data) ;
    tmp → 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 not possible.\n");
      return ;
}
struct node * temp , * prev ;
 temp = head ;
while (temp -> next != NULL) {
      prev -> next = NULL ;
}
   if (temp == head ) {
       head = NULL ;
    } else {
          prev -> next = NULL ;
}
printf ("Element deleted from the end : %d \n", temp ->
   data );
   free (temp);
}
void deleted at position () {
```

```
int position;
printf ("Enter the position to delete :");
if (head == NULL) {
    printf ("list is empty. deletion not possible.\n");
    return;
}
for (int i=0; temp != NULL && i<position; i++)
    {
        prev = temp;
        temp = temp → next;
    printf ("Element at position %d deleted successfully.
            \n", position);
        free (temp);
        return;
    }
for (temp == NULL) {
        print ("position %d is out of bounds. \n",
position);
    return;
}
prev → next = temp → next;
printf ("Element at position %d deleted successfully.
  \n", position);
  free (temp);
}
int main() {
    int choice;
    while (1) {
Printf ("\n 1. to insert at beginning \n
        2. to delete beginning \n
        3. to delete at any position \n.
```

4. to delete at any position
5. to display \n
6. to exit \n");
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 \n");
break;
}
}
return 0;
}

* 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 : 2

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 : 2

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 : 4.

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

1. Enter your choice : 5
Elements are 2, 2, 4, 5.

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 : 2
Element deted from the beginning : 5

1. to insert at the begging.
2.
3.
4.
5.
6.

Enter your choice : 3
Element deleted from the end : 2

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

Enter your choice : 4
Elemelent to delete at any point : 2

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

Leet code :

```
struct listNode* reverseBetween (struct listNode *start
int a, int b)
{
  a -= 1
  b -= 1;
struct list node *node1 = NULL, *node2 = NULL
      * node b = NULL, * node a = NULL, *ptr = start;
int c = 0;
while (ptr != NULL)
  {
  if (c == a - 1)
     node b = ptr ;
  else if (c == a)
     node 1 = ptr ;
  else if (c == b)
     node 2 = ptr;
  else if (c == b+1)
     {
       nodea = ptr ;
       break;
     }
   c+1 = 1 ;
  ptr = ptr→ next ;
}
struct listNode * pre = nodea, * temp;
  ptr = start ;
  c = 0 ;
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 = ptr ;
      else
          node b -> next = ptr ;
      break ;
  }
   else
      ptr = ptr -> next ;
      c+ = 1
  }
   retur start ;
  }
```

→ Output :-

Input

head =

[1, 2, 3, 4, 5]

left = 2

right = 4

output : [1, 4, 3, 2, 5]

Expected : [1, 4, 3, 2 5]

25/01/24

LAB- 06 :-

1. WAP to implement single linked list with following operations : sort the linked list, Reverse the linked list, concatenation of two linked lists.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node * next;
};
struct Node* createNode (int value) {
    struct Node* newnode = (strot Node*) Malloc
    (size of (struct Node));
newNode → data = value;
new Node → Next = NULL;
return newNode;
}
Void insertEnd (struct Node** head, int value) {
    struct Node* new Node = create Node (value);
    if (* head == NULL) {
        * Head = new Node;
} else {
    struct Node* temp = * head;
    while (temp != NULL) {
    printf ("%d → ", temp → data);
    temp = temp → next;
    }
    print f ("NULL \n");
}
void sortlinked list (struct Node * Head) {
    int swapped, i;
    struct Node * ptr;
```

```
struct Node* lptr = 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 concatenateLinkedLists (struct Node ** list1,
    Node* list2) {
            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
    list 1:");
    scanf("%d", &n);
    printf("Enter the elements for list 1:\n");
    for (int i=0; i<n; i++) {
        scanf("%d", &value);
        insertEnd(&list1, value);
    }
    printf("Enter the number of elements for list 2:");
    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:");
    display(list1);
    list2 = reverseLinkedList(list2);
```

```
printf("Reversed list:\n");
printf("Reversed list 2: ");
display(list2);
concatenateLinkedLists(list1, list2);
printf("concatenated list :");
display(list1);
struct Node* temp;
while (list1 != NULL) {
    temp = list1;
    list1 = list1 -> next;
    free(temp);
}
return 0;
}
```

→ output:

Enter the number of elements for list 1: 5
Enter the element for list 1: 1 2 3 4 5

Enter the number of elements for list 2: 5
Enter the elements for list 2: 2 3 4 5 6

sorted list1: 1 → 2 → 3 → 4 → 5 → NULL
Reversed list 2: 6 → 5 → 4 → 3 → 2 → NULL
concatenated list: 1 → 2 → 3 → 4 → 5 → 6 → 5 → 4 → 3 → 2 → 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
    (size of (struct Node));
    new Node → data = value;
    NewNode → next = null;
    return newNode;
}
void push (struct Node ** top, int value) {
    struct Node * NewNode = createNode (value);
    newNode → next = * top;
    * top = new Node;
}
int pop (struct node ** top) {
    if (*top == null) {
        printf ("stack underflow!, \n");
        return -1
    }
    struct Node * temp = * top;
    int popped value = temp → data;
    * top = temp → next;
    free (temp);
    return popped value;
}
```

```
void displayStack (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 ("\n Stack operations:\n");
    printf ("1. Push \n");
    printf ("2. Pop \n");
    printf ("3. Display \n");
    printf ("4. Exit \n");
    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", v
    }
    break;
```

```
case 3:
    display stack (top);
    break;
case 4:
    printf ("Exiting the program. \n");
    break;
default:
    printf(" Invalid choice ! please enter a valid
    option. \n");
}
} while (choice != 4);
struct Node * temp;
while (top != NULL) {
    temp = top;
    top = top → next;
    free (temp);
}
return 0;
}
```

→ output:

```
stack operation
1. push
2. pop
3. display
4. Exit
Enter your choice: 2
Popped value : 2
```

3. Queue:

```
#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 queue* queue, 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 queue * queue) {
    if (queue → front == NULL) {
    printf ("queue underflow! \n");
        return -1;
}
struct Node * temp = queue → front;
int dequeuedvalue = temp → data;
queue → front = temp → next;
if (que → front == NULL) {
    queue → rear = NULL);
}
free (temp);
return dequeed value;
}
void display queue (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 = create queue();
int choice, value;
do {
printf ("\n queue operations : \n");
printf (" 1. Enqueue \n");
```

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

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

→ output:

```
queue operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice : 1
Enter the value to enqueue : 4
```

LAB - 07 :-

Q. WAP to implement doubly link list with primitive operations
a) create a doubly linked list
b) insert new node to the left of the node
c) Delete the node based on a specific value
Display the contents of the list

→
```
#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
(size of (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 be inserted :");
 scanf ("%d", &temp → data);
printf ("Enter the value to the left of which the
     node has to be inserted :");
 scanf ("%d", &key);
 struct node *ptr = start;
```

```
while (ptr != NULL && ptr -> data != key) {
    ptr = ptr -> next;
}
if (ptr == NULL) {
    printf ("Node with value %d not found \n", key);
    free (temp);
}
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 the value to be deleted :");
    scanf ("%d", &value);
    struct node *ptr = start;
    while (ptr != NULL && ptr -> data != value){
        ptr = ptr -> next;
    }
```

```
if (ptr->next != NULL) {
        ptr->next->prev = ptr->prev;
    }
printf ("Node with value %d deleted \n", value);
    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 \n", ptr->data);
        ptr = ptr->next;
    }
  }
}
struct node * insert_right (struct node * start) {
    int value;
    struct node * temp = create_node (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;
    }
```

```
return start;
}
int main() {
    int choice
    while (1) {
printf ("\n1. create a doubly linked list.
        \n2. Insert to the left of a node
        \n3. Delete base on a specific value in..
        Display the contents\n5.
          Insert to right\n6. Exit");
    scanf ("%d", &choice);
switch (choice) {
      case 1:
        s1 = createNode (0);
      printf("Doubly linked list created \n");
        break;
      case 2:
      s1 = insert_left(s1);
      break;
      case 3:
        s1 = delete_value (s1);
      break;
      case 4:
        display (s1);
        break;
      case 6:
        printf (" Exiting the program\n");
        exit (0);
        default:
          printf ("Invalid choice \n");
    }
}
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 the contents
5. Insert to right
6. Exit
1. Doubly linked list created

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

5

Enter value to be inserted 12

<options>

5

Enter value to be inserted 13

<options>

2

Enter value to the inserted : 6

Enter the value to the left of which the node has to be inserted : 13

<options>

5

Enter value to be inserted : 14

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

Leetcode:

<option> 4
list contents : 0 12 6 14

```
# 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 (int i=0; i<k; i++) {
   struct ListNode * part_head = current;
int part_length = part_size + (i<extra_nodes
  ? 1:0);
```

```
for (int i=0; i<partLength-1 && current; i++)
{
    current = current->next;
}
if (current){
    struct ListNode* nextNode = current->next;
    current->next = NULL;
    result[i] = partHead;
    current = nextNode;
} else {
    result[i] = NULL;
}
}
*returnSize = k;
return result;
}
```

⇒ output

Input

head = [1, 2, 3]

k = 5

Output

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

Expected

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

16/02/24

LAB 08:

LAP
a. To construct a binary search tree
b. To traverse the tree using all methods i.e. in-order, preorder and postorder
c. To display the elements in the 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 TreeNode));
newNode -> data = data;
newNode -> left = newNode -> right = NULL;
return new node;
}
struct TreeNode* insertNode(struct TreeNode*
root, int data) {
    if (root == NULL) {
    return createNode(data);
    }
    if (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);
        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 ("In-order traversal:");
    inorderTraversal (root);
    printf ("\n");
    printf ("pre-order traversal:");
    printf ("\n");
    printf ("post-order traversal:");
    printf ("\n");
}

int main () {
    struct TreeNode* root = NULL;
    int choice, data;
    do {
        printf ("1. Insert a node\n");
        printf ("2. Display tree\n");
        printf ("3. Exit\n");
        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.\n")
        } else {
            displayTree(root);
        }
        break;
    case 3:
        printf("Exiting program.\n");
        break;
    default:
        printf("Invalid choice. Please try again.\n");
}
} while (choice != 3);
    return 0;
}
```

→ output:

1. Insert a node
2. Display tree
3. ~~tree~~ Exit

Enter your choice: 1

Enter data to insert: 2

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

Enter your choice : 1
Enter data to insert : 3

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

Enter your choice : 1
Enter data to insert : 4

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

Enter your choice : 2
In-order traversal : 2 3 4 5
Pre-order traversal : 2 3 4 5
post-order traversal : 5 4 3 2

Leet code :

```
struct listnode *rotate right (struct list 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 -> 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]

LAB-09 :-

Date 22/02/24
Page

1. Write a program to traverse a graph using BFS Method

```
#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_destory(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_VERTICES];
    for(int i=0; i<g->v; i++) {
    }
```

```
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]) {
 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 ("Enter the number 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);
   Graph_addEdge (g, v, w);
}
int startvertex;
printf ("Enter the starting vertex for BFS:");
 scanf ("%d", &startvertex);
printf ("Following is Breadth first Traversal (starting
        from vertex %d) \n", startvertex);
 Graph_destroy (g);
  return 0;
}
```

* output:

Enter the number of vertices in the graph: 4
Enter the number of edges in the graph: 6
Enter the edges (v1 v2):
0 1
0 2
1 2
2 0
2 3
3 3
Enter the starting vertex for BFS: 2
Following is Breadth first Traversal (starting from vertex 2)
2 0 3 1

write a program to check whether given graph is connected or not using DFS method :-

→
```
# include <stdio.h>
# include <stdlib.h>
# define MAX_NODES 100
# 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 is connected (int n) {
    DFS (
```

```
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:\n");
    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;
}
```

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

Hackerrank code:

```
#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) {

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) return;
    if (level % k == 0) {
        Node *temp = root->left;
        root->left = root->right;
        root->right = temp;
```

output:

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

LAB prgm 10 :-

Q: lab prgm - Given file of N employee

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define TABLE_SIZE 100
#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;
}
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 file \n");
    return 1;
  }
while (fgets (line, size of (line), file))
  emp = (struct Employee*) malloc (size of (struct
      Employee));
  sscanf (line, "%d %s %f", emp → key,
  emp → name, emp → designation, & emp → salary);
    insert (ht, emp);
  }
  fclose (file);
  printf ("Enter the key to search:");
   scanf ("%d", key);
   emp = search (ht, key);
   if (emp != NULL) {
  printf ("Employee record found at key %d:
      \n" 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 %d\n", key);
  }
  for (int i=0; i < TABLE_SIZE; i++) {
      if (ht.table[i] != NULL) {
       free (ht.table[i]);
      }
  }
  return 0;
}
```

→ Output :

Enter the file name containing employee records
= employees.txt

Enter the key to search : 5678

Employee record found with key 5678 :

Name : Jane

Designation : Developer

Salary : 60000.00