INDEX

USN - 1BM22CS18C

NAME: Nithya Lakshmi.V STD.: ____ SEC.: ____ ROLL NO.: ____ SUB.: DS

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1.	21/12/23	LAB PROGRAMS 1	10	
2.	21/12/23	Practice programs	10	
3.	28/12/23	LAB - 2		
4.	28/12/23	LAB - 3		
5.	11/1/24	LAB - 3 (Singly Linked List)	10	
6.	18/1/24	Leet Code : 1	5	
7.	18/1/24	LAB program 5	10	
8.	18/1/24	Leet code : 2	10	
9.	25/1/24	Sorting, Reverse, Concatenating	10	
10.	25/1/24	Stack & Queue implementations	10	
11.	1/2/24	Doubly linked list	10	
12.	1/2/24	Leet code : 3	10	
13.	15/2/24	Binary Search Tree	10	
14.	15/2/24	Leet code : 4	10	
15.	22/2/24	BFS & DFS	10	
16.	22/2/24	HackerRank	10	
17.	29/2/24	LAB - 10	10	

Swapping 2 numbers 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
20
Before swapping : num1=10 and num2=20
After swapping : num1=20 and num2=10

Write a program to implement Dynamic memory Allocation like MALLOC, CALLOC, FREE and REALLOC

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

void * myMalloc(size_t size)
{
    return realloc(ptr, size);

}
 void* myCalloc (size_t num, size_t size){

return calloc(num, size);
}
 void myfree(void * ptr)
{
    free (ptr);
}

int main()
{
    int * arr1, *arr2;
    size_t size;
    printf("Enter the size of the array\n");
    scanf("%zu", &size);

    arr1 = (int*) myMalloc(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++)
{
    printf("%d ", arr1[i]);
}
printf("\n");

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

if(arr2==NULL)
{
    printf("Memory reallocation failed\n");
    myfree(arr1);
    return 1;

printf("Enter additional elements of the
        array:\n");
for (size_t i=size/2; i<size; i++)
{
    printf("Elements %zu: ", i+1);
    scanf("%d", &arr2[i]);
}
printf("\n");

myfree(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 additional 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
```

Program 3: Stack

```
#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 %d.\n", value);

    }else
    {
        stack -> top++;
        stack -> items[stack -> top] = value;
        printf("Pushed %d onto the
                stack\n", value);
```

```
    }
}
int pop (struct Stack * stack)
{
    int poppedValue = -1;
    if (isEmpty(stack))
    {
        printf("Stack underflow");
    }
    else {

        poppedValue = stack -> items[stack
                                -> top];
        stack -> top--;
        printf("Popped %d from the
                stack \n", poppedvalue);
    }

    return poppedvalue;
}

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, 10);
    push (& stack, 20);
    push (& stack, 30);

    pop (& stack);
    display (& stack);

    push (& stack, 40);
    display (& stack);

    return 0;
}
```

Pushed 10 onto the stack
Pushed 20 onto the stack
pushed 30 onto the stack
Elements in the stack : 10 20 30
popped 30 from the stack
Elements in the stack : 10 20

21/12/23

Practice Programs outputs

Program - 1

Enter name
Nithya
~~12345~~ Enter account number
12345
Enter amount
1000
Enter amount to be withdrawed
500
Remaining balance = 500
Enter amount to be deposited
500
Balance = 1000

Program - 2

Enter 5 words : walk
talk
run
swim
play

In the lexicographical order:
play
run
swim
talk
walk

Program - 3

Array[0][0] = 1
Array[0][1] = 2
Array[0][2] = 3

Array[1][0] = 4
Array[1][1] = 5
Array[1][2] = 6

Array[2][0] = 8
Array[2][1] = 9
Array[2][2] = 7

Enter the number you want :> 7
Number exists

Program - 4

'The world is amazing' contains 'amazing'

Program - 5

Enter the size of the array : 5
Enter the elements of the array:
Element 1 : 1
Element 2 : 2
Element 3 : 3
Element 4 : 4
Element 5 : 5
Enter the number to find the last occurrence : 2
The last occurrence of 2 is at index 1

Program-6

Enter the element to search : 90
Element 90 found at index 4

Program-7

Enter number of elements in the array
6
Enter the array elements
1
4
5
7
8
11
2
Enter value to be searched
5
5 is present at location 2

Program-8

Enter size of the array : 10
Enter elements in the array:
1
2
3
4
6
7
11
2
14
6
8

Maximum element = 14
Minimum element = 1

CamScanner

LAB-2

Program 1: Infix to Postfix Expression

```
#include <stdio.h>
#include <ctype.h>
#define SIZE 50

char stack[SIZE];
int top = -1;
push(char ele)
{
    stack[++top] = ele;
}
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, ele;
    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();
                        ele=pop();
                }
                else
                {
                    while(pr(stack[top])>=pr(
                    postfix[k++]=pop();
                    push(ch);
                }

        }

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

output:

Enter the infix expression : A*B+C*D-E

Postfix expression = AB* CD*+E-

Program 2: Evaluation of Postfix expression

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

output:

Enter the expression : 45*32-

The result of expression 45*32-$ = -1

LAB-3

Program 1: Queue

```
#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 empty \n");
        for (i = front; i <= rear; i++)
            printf("%d", queue_array[i]);
        printf("\n");
    }

}

insert()
{
    int add_item;
    if (rear == MAX-1)
        printf("Queue Overflow\n");
    else
    {
        if (front == -1)
            front = 0;
        printf("Insert the element in
                queue: ");
        scanf("%d", &add_item);
```

```
        rear = rear + 1;
        queue_array[rear] = add_item;
    }
}

delete()
{
    if(front==-1||front>rear)
    {
        printf("Queue Underflow\n");
        return;
    }
    else
    {
        printf("Delete Elements is : %d\n",
                                queue_array[front]);
        front = front+1;
    }
}

main()
{
    int choice;
    while(1)
    {
        printf(" 1. Insert \n");
        printf(" 2. Delete \n");
        printf(" 3. Display \n");
        printf(" 4. Exit \n");
        printf(" Enter your choice : ");
        scanf(" %d", &choice);
        switch(choice)
        {
            case 1:
```

```
insert();
break;
case 2:
 delete();
 break;
case 3:
 display();
 break;
case 4:
 exit(1);
 default:
 printf("Invalid choice\n");
}
}
}
```

output:

```
1. insert
2. delete
3. display
4. exit
enter your choice: 1
insert the element in the queue: 19
1. insert
2. delete
3. display
4. exit
Enter your choice: 3
queue is:
 19
1. insert
2. delete
3. display
4. Exit
```

enter your choice : 4

Program 2: Circular Queue

```
# include <stdio.h>
# define SIZE 5
int items[SIZE]
int front = -1, rear = -1;
int isFull()
{
    if ((front == rear + 1) || (front == 0
                                && rear ==
                                SIZE - 1))

    return 1;
    return 0;
}
int isEmpty()
{
    if (front == -1)
    return 1;
    return 0;
}
void enQueue(int element)
{
    if (isFull())
    printf("\n Queue is full");
    else{

        if (front == -1)
            front = 0;
        rear = (rear + 1) % SIZE;
        items[rear] = element;
        printf("\n Inserted -> %d", element);
    }
}
```

```
int deQueue()
{
    int element;
    if (isEmpty())
    {
        ~~int element;~~
        printf("\n Queue is empty");
        return -1;
    }
    else
    {
        element = items[front];
        if (front == rear)
        {
            front = -1;
            rear = -1;
        }
        else
        {
            front = (front + 1) % SIZE;
        }
        printf("\n Deleted element -> %d\n", element);
        return element;
    }
}

void display()
{
    int i;
    if (isEmpty())
    {
        printf("\n Empty Queue \n");
```

```
else
{
    printf("\n front -> %d ", front);
    printf("\n Items -> ");
    for( i = front ; i != rear ; i = (i+1)
                                    % SIZE)
    {
        printf("%d", item[i]);
    }
    printf("%d", items[i]);
    printf("\n Rear -> %d \n", rear);
}
}

int main()
{
    enQueue(1);
    enQueue(2);
    enQueue(3);
    enQueue(4);
    enQueue(5);
    display();
    deQueue();
    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

Front → 2
Items → 3 4 5
Rear → 4

Inserted → 6
Inserted → 7
Front → 2
Items → 3 4 5 6 7
Rear → 7

Sneha
28/12/23

11/1/24

LAB-3 (Singly 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 isempty\n");
        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 node));
    printf("Enter the value to be inserted :");
    temp -> next = head;
    head = temp;
}
```

```
void insert_end()
{
    struct node * temp, *ptr;
    temp = (struct node *) malloc (sizeof
                                    (struct
                                    node));

    printf(" Enter the values 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 ", & 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;
}
}
int main()
{
  int choice;
  while(1)
  {
    printf("\n1. Insert at the
            beginning\n2.
            Insert at the end\n3.
            Insert at any position\n4.
            Display\n5.
            Exit\n");
```

```
printf("Enter your choice : ");
  scanf("%d",&choice);
  ~~scanf("%~~

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");
  }
}
return0;
}
```

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 inserted : 19

11/1/24

classmate

Date
Page

Leetcode (Stack program)

```
#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)*10000);
    stack->minStack = (int*)malloc(sizeof(int)
                                    *10000);

    stack->top = -1;
    return stack;
}

void minStackPush(MinStack * obj, int val){
    obj->top++;
    obj->stack[obj->top] = val.
    if (obj->top == 0 || val <= obj->minStack
                                [obj->top-1])

    {
        obj->minStack[obj->top] = val;
    }else
    {
        obj->minStack[obj->top] = obj->minStack
                                [obj->top-1];
```

```
}
}

void minStackPop(MinStack * obj)
{
    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);
}
```

classmate

Date
Page

18/1/24

LAB Program - 5 (Delete and display implementation of Singly Linked List)

```
#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
                        (structnode));
    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(" Elements 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 = 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.\n");
        return;
    }
    struct node *temp, *prev;
    temp= head;
    if(position == 0)
    {
        head = head -> next;
        printf("Elements 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 successfully \n", Position);
  free(temp);
}

int main()
{
    int choice;
    while (1)
    {
        printf("\n 1. to insert at the beginning \n 2. to delete at beginning \n 3. to delete at end \n 4. to delete at any position \n 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);
}
```

Output :-

1. to insert at the beginning
2. to delete at 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 : 19

18/1/24

classmate

Date
Page

18/1/24

LEET CODE (Reverse Linked List)

```
struct ListNode* reverseBetween(struct
                                listNode *start,
                                int a, int b)
{
  a = -1;
  b = -1;
  struct ListNode* node1 = NULL, *node2 =
                                NULL *node3 =
                                NULL,
                                *Nodea = NULL
                                *ptr = start;
  int c = 0;
  while(ptr != NULL)
  {
    if (c == a-1)
     node3 = 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;
```

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

18/1/24

classmate

Date
Page

25/1/24

Program-1

```
#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 Nod

    newNode -> data = value;
    newNode -> next = NULL;
    return newNode;
}
void insertEnd(Struct Node** head, int
                                value)
{
    Struct Node * newNode = CreateNode(valu
    if(*head ==NULL){
      * head = newNode;
    } else {
        Struct Node* temp = * head;
        while (temp -> next!=NULL)
        {
            temp = temp -> next;
        }
    }
void display(Struct Node* head)
{
    Struct Node* temp = head;
```

```
while (temp != NULL)
{
    printf("%d -> ", temp -> data);
    temp = temp -> next;
}
printf("NULL\n");
}
~~void sortLinke~~

void sortLinkedlist(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 = NULL;
    while (current != NULL){
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    return prev;
}
void concatenateLinkedLists(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;
    }
}
```

```
int main()
{
    StructNode * list2 = NULL;
    StructNode * list1 = 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 : ");
    for (int i = 0; i < n; i++)
    {
        scanf(" %d ", &value);
        insertEnd(&list1, value);
    }

    scanf(" %d ", &n);
    printf(" Enter the elements for list2
            : \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 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: 1
Enter the elements of for list 1:
9
Enter the number of elements for list 2: 2
Enter the elements for list 2:
19
28
Sorted list 1: 9 -> NULL
Reversed list 2: 28 -> 19 -> NULL
Concatenated list: 9 -> 28 -> 19 -> NULL
```

Program-2 (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));
    newNode->data = 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");
        return -1;
    }
```

```
StructNode *temp = *top;
int poppedvalue = temp->data;
*top = temp->next;
free(temp);
return poppedvalue;
}

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 Operation \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:
        ~~pop~~ 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 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 Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice =1
Enter the value to push =19
```

CamScanner

Program-2 (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");
        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 dequeueValue;
}
```

```
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 = createQueue();
    int choice, value;
    do{
        printf("Queue Operations:");
        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("Dequeue value : %d\n",
                                value);
    }
    break;
case 3;
    display(Queue (queue);
    break;
case 4;
    printf(" Exiting the program!\n");
    break;
case
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(temp);
}
free(queue);
return 0;
```

}

output:

Queue operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice : 1
Enter the value to enqueue : 19

25/1/24

1/2/24

Program - 1 (Doubly linked list with primitive operations)

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

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

```
int main()
{
    int choice;
    while (1) {
        printf("\n1. Create a doubly linked
                list\n2. Insert to the
                left of a node\n3.
                Delete based on
                a specified value
                \n4. Display
                the contents
                \n5. Exit\n");

    scanf("%d", &choice);
    switch (choice)
    {
        case 1:
            s1 = createNode(0);
            printf(" Doubly 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:
            printf(" Existing 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. Exit

1

Doubly linked list created

~~LEETCOD~~

LEET CODE (Split linked list in parts)

```
#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 * sizeof
                                 (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 j = 0; j < part_length - 1 && current;
                                       j++)
        {
            current = current -> next;
        }
```

```
if (current){
    struct ListNode* next_node = current->next;
    current->next = NULL;
    result[i] = part_head;
    current = next_node;
} else {
    result[i] = NULL;
}
* return size = k;
return result;
}
```

<u>Output</u>: head = [1, 2, 3]
k = 5
[[1], [2], [3], [], []]

8/2/24

5/2/24

LAB-8 (Binary Search 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 newNode;
}
struct TreeNode* insertNode (struct TreeNode
                                *root, int data)
{
    if (root == NULL)
    {
        return createNode(data);
    }
    if (data < root -> data)
    {
        root -> left = insertNode (root->left, data);
    }
    return root;
}
```

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

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("in-order Traversal:");
    inOrderTraversal(root);
    printf("\n");
```

```
printf(" Pre - Order Transversal");
preOrderTraversal(root);
printf("\n");
printf(" Post-order traversal");
postOrderTraversal(root);
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. Exit
Enter your choice : 1
Enter data to insert : 50
```

15/7/24

LEET CODE : Rotate list

Source code:

```
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;
    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;
}
```

<u>Output</u>

case1

head = [1, 2, 3, 4, 5]
k = 2

[4, 5, 1, 2, 3]

head = [0, 1, 2]
k = 4

[2, 0, 1]

16/2/24

classmate

Date
Page

2/12/24

LAB-9 (BFS) Program-1

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

typedef struct {
    int matrix[MAX_VERTICES][MAX_VERTICES];
    int numVertices;
} Graph;

Graph* createGraph(int numVertices)
{
    Graph* graph = (Graph*) malloc(sizeof(Graph));
    graph -> numVertices = numVertices;

    for(int i = 0; i< numVertices; i++)
    {
        for( int j = 0; j< numVertices; j++)
        {
            graph -> matrix[i][j] = 0;
        }
    }
    return graph;
}
void addEdge(Graph* graph, int src, int dest)
{
    graph -> matrix[src][dest] = 1;
}
```

2/12/24

```
Void bfs (Graph* graph, int startVertex)
{
    int visited[MAX_VERTICES]={0};
```

```
int queue[MAX_VERTICES];
int front = 0, rear = 0;
queue[rear++] = startVertex;
visited[startVertex] = 1;

printf("BFS Traversal starting from
        vertex %d:", startVertex);
while(front < rear)
{
    int currentVertex = queue[front++];
    printf("%d", currentVertex);

for(int i = 0; i < graph->numVertices; i++)
{
    if (graph->matrix[currentVertex][i]
                && !visited[i])
    {
        queue[rear++] = i;
        visited[i] = 1;
    }
}
}
printf("\n");
}

int main()
{
    int numVertices, numEdges, src, dest,
        startVertex;

    printf("Enter the number of vertices
        in the graph:");
    scanf("%d", &numVertices);
```

```
Graph* graph = CreateGraph(numVertices);

printf("Enter the number of edges in the graph:");
scanf("%d", &numEdges);

printf("Enter the edges (source destination):\n");
for(int i = 0; i < numEdges; i++)
{
    scanf("%d %d", &src, &dest);
    addEdge(graph, src, dest);
}

printf("Enter the starting vertex for BFS traversal:");

scanf("%d", &startVertex);

bfs(graph, startVertex);

return(0);
}
```

output:

```
Enter the number of vertices in the graph: 6
Enter the number of edges in the graph: 7
Enter the edges (source destination):
0 1
0 2
1 2
1 3
2 3
```

34
45
Enter the starting vertex for BFS traversal:
BFS traversal starting from vertex 0 : 0 1 2 3 4 5

Program - 2 (DFS)

```
#include <stdio.h>
# include < stdlib.h>
struct AdjListNode{

    int dest;
    struct AdjListNode * next;
};
struct AdjList{
    struct AdjListNode * head;
};
struct Graph{
    int V;
    struct AdjList * array;
};

struct AdjListNode* newAdjListNode(int dest)
{
    struct AdjNode *newNode = (struct AdjNode*) malloc(sizeof (struct AdjList Node));

    newNode -> dest = dest;
    return newNode;
}
struct Graph* createGraph.(int V)
{
```

```
struct Graph *graph = (struct Graph*)
                malloc(sizeof(struct
                Graph));
graph->V = V;

graph->array = (struct AdjList*) malloc
                (V * sizeof(struct
                AdjList));

for(int i = 0; i < V; ++i)

    graph->array[i].head = NULL;
    return graph;
}
void addEdge (struct Graph* graph, int src,
                int dest){
    struct AdjListNode * newNode = newAdjList
                Node(dest);
    newNode->next = graph->array[src].head;
    graph->array[src].head = newNode;

    newNode = newAdjListNode(src);
    newNode->next = graph->array[dest].head;
    graph->array[dest].head = newNode;
}

void DFSUtil(struct Graph * graph, int v,
                int * visited){
    visited[v] = 1;
    struct AdjList Node * temp = graph->array[v]
                .head;
    while(temp != NULL){
        int adjVertex = temp->dest;
        if(!visited[adjVertex])
            DFSUtil(graph, adjVertex, visited);
```

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

int isConnected(struct Graph* graph, int V)
{
    int* visited = (int*) malloc(V* sizeof(int));
    int i;
    for(i=0; i<V; ++i)
    if (graph -> array [i].head! =NULL)
    {
        DFSUtil(graph, i, visited);
        break;
    }
    for (i=0; i<V; ++i)
        if (visited [i] ==0)
            return 0;
        return 1;
}

int main()
{
    int V, E;
    printf("Enter the number of vertices:")
    scanf("%d", &V);
    struct Graph * graph = CreateGraph(V);
    printf("Enter the number of edges:").
    scanf("%d", &E);
    printf("Enter of edges (src dest):\n")
    for( int i=0; i<E; ++i)
    {
        int src, dest;
        scanf("%d%d", &src, &dest);
        addEdge (graph, src, dest);
    }
    if (isConnected (graph, V))
```

```
    printf("The graph is connected.\n");
    else
    printf("The graph is not connected\n");
    return 0;
}
```

Output:

```
Enter the number of vertices : 5
Enter the number of edges : 4
Enter edges (src dest):
01
02
03
04
The graph is connected.
```

24/12/24

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) return;
        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);
}

int** swapNodes(int indexes_rows, int indexes_
                columns, int** indexes,
                int queries_count, int*
                queries, int* result_row
                int* result_columns)
{
    Node** nodes = (Node**) malloc((indexes_
                                    row+1)*
    for(int i=1; i<=indexes_rows;i++) sizeof(Node*));
    {
        nodes[i]=createNode(i);
int main()
{
    int n;
    scanf("%d",&n);
    int** indexes= malloc(n*sizeof(int**));
    for(int i=0; i<n;i++){
     indexes[i] =malloc(2* sizeof(int));
    for (int i=0; i<n; i++){
     free(indexes[i]);
    }
    free(indexes);
    free(queries);
    return 0;
}
```

OUTPUT:

input:
3
2 3
-1 -1
-1 -1
2
1
1

output:
3 1 2
2 1 3

29/2/24 LAB-10 (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;
  }
  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, sizeof(line), file)){
    emp = (struct Employee*) malloc(sizeof(struct
                                        Employee));
    if (emp != NULL){
        printf("Employee record found with key %d:", emp
        printf("Name: %s\n", emp->name);
        printf("Designation: %.2f\n", emp->designation
        printf("Salary: %.2f\n", emp->salary);
}else{
    printf("Employee record not found for key
                    %s\n", key);
}

return 0;
}
```

Output:

```
Enter the filename containing employee records: employee.
Enter the key to search: 1234
Employee record found with key 1234:
Name: Nithya
Designation: manager
Salary: 10000.00
```