INDEX

NAME: Rahul. N. Raju STD.: III sem SEC.: _____ 1BM22CS215 ROLL NO.: _____ SUB.: Data Structures

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1	21/12/23	Lab Program - 1		
2	28/12/23	Lab Program - 2		
3	11/1/24	Lab Program - 3		
4	11/1/24	Lab Program - 4		
5	18/1/024	Leetcode Problem - 1		
6	18/1/24	Lab Program - 5		
7	1/2/24	~~Lab Program - 6~~ Leetcode 2		
8	25/1/24	Lab Program - 6		
9	1/2/24	Lab Program - 7		
10	1/2/24	Leetcode Problem - 3		
11	15/2/24	Lab Program - 8		
12	15/2/24	Leetcode Problem - 4		
13	22/2/24	Lab Program - 9		
14	29/2/24	Hackerrank Problem		
15	29/2/24	Lab Program - 10		

Lab - 1

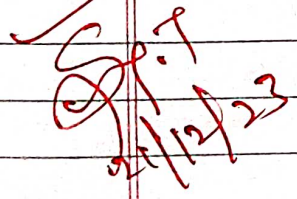
3) Stack Implementation

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 4
int top = -1;
int a[SIZE];
void push();
void pop();
void show();

void main()
{
    int ch;
    while (1)
    {
        printf("Operations on the stack: \n");
        printf("1. Push the element 2. Pop the element 3. Show 4. End \n");
        printf("Enter the choice: \n");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1: push();
                    break;
            case 2: pop();
                    break;
            case 3: show();
                    break;
            case 4: exit(0);
            default: printf("Invalid choice");
        }
    }
}
```

```
void push ()
{
    int x;
    if (top == SIZE - 1)
        printf ("Overflow \n");
    else
    {
        printf ("Enter the element to be added: \n");
        scanf ("%d", &x);
        top = top + 1;
        a[top] = x;
    }
}
void pop ()
{
    if (top == -1)
        printf ("Underflow \n");
    else
    {
        printf ("Popped element : %d \n", a[top]);
        top = top - 1;
    }
}
void show ()
{
    if (top == -1)
        printf ("Underflow \n");
    else
    {
        printf ("Elements in the stack are: \n");
        for (int i = top ; i >= 0; i--)
            printf ("%d \n", a[i]);
    }
}
```

Output:

```
Operations on the stack:
1. Push the element 2. Pop the element 3. show 4. End
Enter the choice: 1
Enter the element to be added: 3
Operations on the stack:
1. Push the element 2. Pop the element 3. Show 4. End
Enter the choice: 1
Enter the element to be added: 2
Operations on the stack:
1. Push the element 2. Pop the element 3. show 4. End
Enter the choice: 2
Popped element: 2
Operations on the stack:
1. Push the element 2. Pop the element 3. show 4. End
Enter the choice: 3
Elements is the stack are: 3
Operations on the stack:
1. Push the element 2. Pop the element 3. show 4. End
Enter the choice: 4
```



28-12-23

Lab - 2

1) Infix to postfix

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 100
char stack[MAX];
int top = -1;
void push(char);
char pop();
int precedence(char);
void infixToPostfix(char infix[], char postfix[]);

void push(char item)
{
    if (top == MAX-1)
        printf("stack overflow \n");
    else
    {
        top++;
        stack[top] = item;
    }
}

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

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

void infixToPostfix (char infix[], char postfix[])
{
    int i=0, j=0;
    char symbol, temp;
    push ('#');
    while ((symbol = infix[i++]) != '\0')
    {
        if (symbol == '(')
            push (symbol);
        else if (isalnum (symbol))
            postfix[j++] = symbol;
        else if (symbol == ')')
        {
            while (stack[top] != '(')
                postfix[j++] = pop ();
            temp = pop ();
        }
        else
        {
            while (precedence (stack[top]) >= precedence (symbol))
                postfix[j++] = pop ();
            push (symbol);
        }
    }
}
```

```
    while (stack[top] != '#')
        postfix[j++] = pop();
    postfix[j] = '\0';
}

int main()
{
    char infix[MAX], postfix[MAX];
    printf("Enter a infix expression: \n");
    scanf("%s", infix);
    infixToPostfix(infix, postfix);
    printf("The postfix expression is: %s \n", postfix);
    return 0;
}
```

Output:

Enter a infix expression:

a*b+c*d-e

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

2) Evaluating the postfix expression

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

int stack[MAX_SIZE];
int top = -1;
void push (int item)
{
    if(top >= MAX_SIZE -1)
        printf("stack overflow");
    top ++;
    stack[top] = item;
}

int pop()
{
    if(top == -1)
        printf("stack underflow");
        return -1;
    int item = stak[top];
    top --;
    return item;
}

int is_operator(char symbol)
{
    if(symbol == '+' || symbol == '-' || symbol == '*' || symbol == '/')
        return 1;
    return 0;
}
```

```
int evaluate (char * expression)
{
    int i=0;
    char symbol = expression[i];
    int operand1, operand2, result;

    while (symbol != '\0')
    {
        if (symbol >= '0' && symbol <= '9')
        {
            int num = symbol - '0';
            push(num);
        }
        else if (is_operand(symbol))
        {
            operand2 = pop();
            operand1 = pop();
            switch(symbol)
            {
                case '+': result = operand1 + operand2; break;
                case '-': result = operand1 - operand2; break;
                case '*': result = operand1 * operand2; break;
                case '/': result = operand1 / operand2; break;
            }
            push(result);
        }
        i++;
        symbol = expression[i];
    }
    result = pop();
    return result;
}
```

```
int main()
{
    char expression[MAX_SIZE];
    printf("enter postfix expression \n");
    scanf("%s", expression);
    int result = evaluate(expression);
    printf("Result = %d \n", result);
    return 0;
}
```

Output :

Enter the postfix expression :

12* 34* +5 -

result = 9

(3a) WAP to simulate the working of a queue of integers using a

```
#include <stdio.h>
#define MAX 50
int queue[MAX];
int rear = -1, front = -1;

void insert()
{
    int num;
    printf("Enter the number to be inserted in the queue : \n");
    scanf("%d", &num);
    if(rear == MAX-1)
        printf("Overflow");
    else if(front == -1 && rear == -1)
        front = rear = 0;
    else
        rear++;
    queue[rear] = num;
}

int delete()
{
    int val;
    if(front == -1 || front > rear)
        printf("Underflow");
        return -1;
    else
    {   val = queue[front];
        front++;
        if(front > rear)
            front = rear = -1;
        return val;
    }
}
```

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

void main()
{
    int option, val;
    do {
        printf("1. Insert\n");
        printf("2. Delete\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your option:\n");
        scanf("%d", &option);
        switch (option)
        {
            case 1: insert();
                    break;
            case 2: val = delete();
                    if(val != -1)
                        printf("the number is deleted : %d", val);
                    break;
            case 3: display();
                    break;
            case 4: exit(0);
        }
    } while (option != 4);
}
```

Output:

1. Insert
2. Delete
3. Display
4. Exit

Enter your option: 1

Enter the number to be inserted in the queue: 10

1. Insert
2. Delete
3. Display
4. Exit

Enter your option: 1

Enter the number to be inserted in the queue: 20

1. Insert
2. Delete
3. Display
4. Exit

Enter your option: 2

~~Enter~~ the number is deleted: 10

1. Insert
2. Delete
3. Display
4. Exit

Enter your option: 3

10

Sol

11-1-24

Week - 3

(3b) WAP to simulate the working of a circular queue of integers using an array

```
#include <stdio.h>
#define SIZE 10
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("Queue is full");
    else {
        if (front == -1)
            front = 0;
        rear = (rear + 1) % SIZE;
        items[rear] = element;
        printf("Inserted %d", element);
    }
}

int dequeue(){
    int element;
    if (isEmpty()) {
        printf("Queue is empty");
        return -1;
```

```
        else
        {
            element = items[front];
            if (front == rear)
                front = -1;
                rear = -1;
        }
    else {
        front = (front+1) % SIZE;
    }
    printf("Deleted element %d ", element);
    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) % SIZE)
            printf("%d", items[i]);
        printf("%d", items[i]);
        printf("Rear = %d", rear);
}


void main()
{
    int option, val;
    do {
        printf("1. Insert\n");
        printf("2. Delete\n");
        printf("3. Display\n");
        printf("4. Exit\n");
```

```
printf("Enter your option: \n");
scanf("%d", &option);
switch (option)
{
    case 1: enqueue();
            break;
    case 2: val = dequeue();
            if(val != -1)
                printf("the number deleted is : %d", val);
            break;
    case 3: display();
            break;
    }
}while (option != 4);
}
```

Output:

1. Insert
2. Delete
3. Display
4. Exit

Enter your option: 1
enter the element : 1
Inserted → 1

1. Insert
2. Delete
3. Display
4. Exit

Enter your option: 2
Deleted element : 1
the number deleted is : 1

1. Insert
2. Delete
3. Display
4. Exit

Enter your option : 1
enter the element : 2
Inserted → 2

1. Insert
2. Delete
3. Display
4. Exit

Enter your option: 3
Front → 1
No. Items → 2
Rear → 1

11-1-24

Week - 4

(Q) WAP to show implementation of insert and display using singly linked list.

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

void insert (struct Node **head, int data)
{
    struct Node *newnode = (struct Node*) malloc (sizeof (struct Nod
    newnode →
    data = data;
    newnode →
    next = *head;
    *head = newnode;
}

void display ( struct Node *node)
{
    printf("\n Linked list: ");
    while (node != NULL)
    {
        printf("%d", node → data );
        node = node → next;
    }
    printf("\n");
}
```

```
void main ()
{
    struct Node *head = NULL;
    insert (&head, 100);
    insert (&head, 80);
    insert (&head, 60);
    insert (&head, 40);
    insert (&head, 20);
    display (head);
}
```

Output:

Linked list : 20 40 60 80 100

18-1-24

Week-5

WAP to delete the first element, specified element and last element and display the linked list.

```
#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 insertAtEnd (struct Node ** head, int value)
{
    struct Node * newNode = createNode (value);
    if (*head == NULL)
        *head = newNode;
    else
    {
        struct Node* temp = *head;
        while (temp -> next != NULL){
            temp = temp -> next; }
        temp -> next = newNode;
    }
}
```

```
void deleteFirst(struct Node** head)
{
    if(*head != NULL) {
        struct Node* temp = *head;
        *head = (*head) -> next;
        free(temp);
    }
}

void deleteEle(struct Node** head, int value)
{
    struct Node* current = *head;
    struct Node* prev = NULL;
    while(current != NULL && current -> data != value)
    {
        prev = current;
        current = current -> next;
    }
    if(current == NULL)
        printf("empty");
    if(prev == NULL)
        *head = current -> next;
    else
        prev -> next = current -> next;
    free(current);
}

void deleteLast(struct Node** head)
{
    if(*head == NULL)
        printf("empty");
    struct Node* temp = *head;
    struct Node* prev = NULL;
```

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

void main()
{
    struct Node* head = NULL;
    insertAtEnd (&head, 1);
    insertAtEnd (&head, 2);
    insertAtEnd (&head, 3);
    insertAtEnd (&head, 4);
    printf (" initial linked list : ");
    display (head);

    deleteFirst (&head);
    printf (" After deleting the first element: ");
    display (head);
```

```
    deleteEle(&head, 2);
    printf("after deleting the specified element: ");
    display(head);

    deleteLast(&head);
    printf("after deleting the last element: ");
    display(head);
}
```

Output:

initial linked list: 1 → 2 → 3 → 4 → NULL
after deleting first element: 2 → 3 → 4 → NULL
after deleting the specified element: 3 → 4 → NULL
after deleting the last element: 3 → NULL

18/11/24

* Leetcode Problems

1) (155) Min Stack

```
typedef struct {
    int str[8000];
    int top;
    int min[1000];
    int minent;
} MinStack;

MinStack * minStackCreate() {
    MinStack *Min;
    Min = (MinStack *) malloc (sizeof (MinStack));
    Min -> top = -1;
    Min -> minent = 0;
    return Min;
}
void minStackPush (MinStack* obj, int x) {
    obj -> top++;
    obj -> str[obj -> top] = x;
    printf ("minent = %d push : %d\n", obj -> minent, x);
    if (obj -> minent == 0 || x <= obj -> min[obj -> minent - 1])
        obj -> min [obj -> minent++] = x;
        printf ("%d*", x);
    printf ("\n === end === \n");
}

void minStackPop (MinStack* obj) {
    if (obj -> top == -1)
        return;
    if (obj -> minent == 0)
        ;
    else if (obj -> str[obj -> top] == obj -> min [obj -> minent - 1])
        obj -> minent--;
    obj -> top--;
}
```

```
int minStackTop (MinStack* obj) {
    return obj -> str [obj -> obj];
}
int minStackGetMin (MinStack *obj) {
    return obj -> min [obj -> minent - 1];
}
Void minStackFree (Minstack * obj) {
    free (obj);
}
```

Output:

["MinStack", "push", "push", "push", "getMin", "pop", "top", "getMin"]
[[], [-2], [0], [-3], [], [], [], []]

25-1-24

Week-6

(a) Linked list operations:- sort, reverse, concatenate

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
    int data;
    struct Node *next;
};
typedef struct Node Node;
Node *createNode (int value)
{
    Node *newNode = (Node*) malloc(sizeof (Node));
    newNode ->data = value;
    newNode -> next = NULL;
    return newNode;
}
void display (Node *head)
{
    while (head != NULL) {
        printf("%d -> ", head -> data);
        head = head -> next;
    }
    printf("NULL \n");
}
Node *sortList (Node *head)
{
    if (head == NULL || head -> next == NULL)
        return head;
    int swapped;
    Node *temp;
    Node *end = NULL;
    do
    {
```

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

Node *reverseList (Node *head)
{
    Node *prev = NULL;
    Node * current = head;
    Node * nextNode = NULL;
    while (current != NULL)
    {
        nextNode = current -> next;
        current -> next = prev;
        prev = current;
        current = nextNode;
    }
    return prev;
}

Node * concatLists (Node *list1, Node *list2)
{
    if (list1 == list2)
        return list2;
```

```
    Node *temp = list1;
    while (temp -> next != NULL)
        temp = temp -> next;
    temp -> next = list2;
    return list1;
}

void main()
{
    Node *list1 = createNode(3);
    list1 -> next = createNode(1);
    list1 -> next -> next = createNode(4);
    Node *list2 = createNode(2);
    list2 -> next = createNode(5);

    printf("original list 1: ");
    display(list1);
    printf("original list 2: ");
    display(list2);
    list1 = sortList(list1);
    printf("sorted list: ");
    display(list1);
    list1 = reverseList(list1);
    printf("reversed List 1: ");
    display(list1);

    Node *concaterated = concatLists(list1, list2);
    printf("concaterated list: ");
    display(concaterated);
}
```

Output:

```
original list 1:    3 -> 1 -> 4 -> NULL
original list 2:    2 -> 5 -> NULL
sorted list 1:      1 -> 3 -> 4 -> NULL
reversed list 1:    4 -> 3 -> 1 -> NULL
concaterated list:  4 -> 3 -> 1 -> 2 -> 5 -> NULL
```

2) Implement stack using linked list

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

struct Node
{
    int data;
    struct Node *next;
};
typedef struct Node Node;
Node *createNode(int value)
{
    Node *newNode = (Node*)malloc(sizeof(Node));
    newNode->data = value;
    newNode->next = NULL;
    return newNode;
}
void display(Node *head)
{
    while(head != NULL)
    {
        printf("%d -> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}
typedef struct{
    Node *top;
} LinkedList;

void push(LinkedList *stack, int value)
{
    Node *newNode = createNode(value);
```

```
    newNode -> next = stack -> top;
    stack -> top = newNode;
}
int pop(LinkedList *stack)
{
    if( stack -> top == NULL)
        printf("stack is empty: ");
        return -1;
    int poppedValue = stack -> top -> data;
    Node * temp = stack -> top;
    stack -> top = stack -> top -> next;
    free(temp);
    return poppedValue;
}

void main()
{
    LinkedList stack;
    stack.top = NULL;
    printf("stack operations: \n");
    push( &stack, 10);
    push (&stack, 20);
    push (&stack, 25);
    push (&stack, 30);
    display ( stack.top );
    printf ("popped value: ", pop(&stack));
    printf ("popped value: ", pop (&stack));
    display (stack.top);
}
```

Output: stack operations:

30 → 25 → 20 → 10 → NULL

popped value: 30

popped value: 25

20 → 10 → NULL

3) Implement queue using Linked list

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

struct Node
{
    int data;
    struct Node *next;
};
typedef struct Node Node;
Node *createNode(int value)
{
    Node *newNode = (Node*) malloc (sizeof(Node));
    newNode -> data = value;
    newNode -> next = NULL;
    return newNode;
}
void display (Node *head)
{
    while (head != NULL)
    {
        printf ("%d -> ", head -> data);
        head = head -> next;
    }
    printf("NULL \n");
}
typedef struct {
    Node *front;
    Node *rear;
} LinkedList;
void enqueue (LinkedList *queue, int value)
{
    Node *newNode = createNode (value);
```

```
    if(queue->front == NULL)
    {
        queue->front = newNode;
        queue->rear = newNode;
    }
    else
    {
        queue->rear->next = newNode;
        queue->rear = newNode;
    }
}
int dequeue(LinkedList *queue)
{
    if(queue->front == NULL)
        printf("queue is empty:\n");
        return -1;
    int dequeuedvalue = queue->front->data;
    Node *temp = queue->front;
    queue->front = queue->front->next;
    free(temp);
    return dequeuedvalue;
}

void main()
{
    LinkedList queue;
    queue.front = NULL;
    queue.rear = NULL;
    printf("\n queue operations:\n");
    enqueue(&queue, 40);
    enqueue(&queue, 50);
    enqueue(&queue, 60);
    display(queue.front);
    printf("dequeued from queue: %d\n", dequeue(&queue));
```

```
    printf("dequeued from queue : %d \n", dequeue(&queue));
    display(queue.front);
}
```

output:

queue operations:

40 → 50 → 60 → NULL

dequeued from enqueue : 40

dequeued from enqueue : 50

60 → NULL

1-2-24

Week-7b

WAP to implement doubly link list with primitive operations

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

struct Node *createNode (int data)
{
    struct Node *newNode = (struct Node*) malloc (sizeof (struct Node
    if (newNode == NULL)
        printf("Memory allocation failed \n");
        exit(1);
    newNode -> data = data
    newNode -> prev = NULL;
    newNode -> next = NULL;
    return newNode;
}
void insertNode (struct Node *head, struct Node *forget, int data)
{
    struct Node *newNode = createNode (data);
    if (forget -> prev != NULL)
    {
        forget -> prev -> next = newNode;
        newNode -> prev = forget -> prev;
    }
    else
        head = newNode;
    newNode -> next = forget;
    forget -> prev = newNode;
}
```

```
void deleteNode (struct Node *head, int value)
{
    struct Node *current = head;
    while (current != NULL)
    {
        if (current -> data == value)
        {
            if(current -> prev != NULL)
                current -> prev -> next = current -> next;
            else
                head = current -> next;
            if (current -> next != NULL)
                current -> next -> prev = current -> prev;
            free (current);
            return;
        }
        current = current -> next;
    }
    printf ("node with value %d not found \n", value);
}

void display (struct Node *head)
{
    printf (" doubly linked list: \n");
    while (head != NULL)
    {
        printf ("%d <-> ", head -> data);
        head = head -> next;
    }
    printf ("NULL \n");
}
```

```
void main()
{
    struct Node *head = NULL;
    head = createNode(1);
    head -> next = createNode(2);
    head -> next -> prev = head;
    head -> next -> next = createNode(3);
    head -> next -> next -> prev = head -> next;
    display(head);
    insertNode(head, head -> next, 10);
    printf("after insertion : \n");
    display(head);

    deleteNode(head, 2);
    printf("after deletion : \n");
    display(head);
}
```

Output:

doubly linked list:

1 ⟷ 2 ⟷ 3 ⟷ NULL

after insertion:

doubly linked list:

1 ⟷ 10 ⟷ 2 ⟷ 3 ⟷ NULL

after deletion:

doubly linked list:

1 ⟷ 10 ⟷ 3 ⟷ NULL

Leetcode - Problem 2 (92)
→ Reverse linked list - II

```
struct ListNode* reverseBetween(struct ListNode* head, int left, int right)
{
    if(head == NULL) return NULL;
    if(left == right) return head;
    struct ListNode * prev = NULL;
    struct ListNode * curr = head;

    int index = 1;
    while (index < left)
    {
        prev = curr;
        curr = curr -> next;
        index++;
    }
    struct ListNode * leftMinusOneNode = prev;
    struct ListNode* leftNode = curr;
    struct ListNode * next = NULL;
    while (left <= right)
    {
        next = curr -> next;
        curr -> next = prev;
        prev = curr;
        curr = next;
        left++;
    }
    if (leftMinusOneNode == NULL)
        head = prev;
    else
        leftMinusOneNode -> next = prev;
    leftNode -> next = curr;
    return head;
}
```

O/P:-
head = [1,2,3,4,5], left = 2, right = 4
output = [1,4,3,2,5]

Leetcode Problem - 3 (725)
(Split linked list in parts)

```
int getLength (struct ListNode* head) {
    int length = 0;
    while (head != NULL) {
        length++;
        head = head -> next;
    }
    return length; }
struct ListNode ** splitListToParts (struct ListNode* head, int k, int* returnSize) {
    int length = getLength (head);
    int partSize = length / k;
    int remainder = length % k;
    struct ListNode ** result = (struct ListNode **) malloc (k * sizeof (struct ListNode*));
    *returnSize = k;
    for (int i = 0; i < k; i++) {
        int currentPartSize = partSize + (i < remainder ? 1 : 0);
        if (currentPartSize == 0)
            result[i] = NULL;
        else {
            result[i] = head;
            for (int j = 0; j < currentPartSize - 1; j++) {
                head = head -> next;
            }
            struct ListNode *temp = head -> next;
            head -> next = NULL;
            head = temp;
        }
    }
    return result;
}
```

Output: head = [1, 2, 3]
k = 5
[[1], [2], [3], [], []]

5-2-24

Week-8

Write a program to construct a binary search tree Traverse and display using inorder, postorder, preorder.

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
    int data;
    struct Node* left;
    struct Node* right;
};
struct Node * newNode(int data)
{
    struct Node *node = (struct Node*)malloc(sizeof(struct Node));
    node -> data = data;
    node -> left = node -> right = NULL;
    return node;
}
struct Node *insert (struct Node *root, int data)
{
    if(root == NULL)
        return newNode(data);
    if(data <= root -> data)
        root -> left = insert(root -> left, data);
    else
        root -> right = insert(root -> right, data);
}
void inorder (struct Node* temp)
{
    if (temp == NULL)
        return;
    inorder (temp -> left);
    printf("%d ", temp -> data);
    inorder (temp -> right);
}
```

```
void preorder (struct Node * temp)
{
    if (temp == NULL)
        return;
    printf ("%d ", temp -> data);
    preorder (temp -> left);
    preorder (temp -> right);
}
void postorder (struct Node * temp)
{
    if (temp == NULL)
        return;
    postorder (temp -> left);
    postorder (temp -> right);
    printf ("%d ", temp -> data);
}

void main()
{
    struct Node *root = NULL;
    int data, choice;
    root = insert (root, 20);
    root = insert (root, 10);
    root = insert (root, 5);
    root = insert (root, 15);
    root = insert (root, 40);
    root = insert (root, 30);
    root = insert (root, 50);
    printf ("\n the inorder traversal is: \n");
    inorder (root);
    printf ("\n");
    printf ("\n the preorder traversal is: \n");
    preorder (root);
    printf ("\n");
```

15/2/24

```
    printf("\n the postorder traversal is : \n");
    postorder(root);
    printf("\n");
}
```

Output :-

the inorder traversal is :

5 10 15 20 30 40 50

the preorder traversal is :

20 10 5 15 40 30 50

the postorder traversal is :

5 15 10 30 50 40 20

Leetcode Problem - 4 (61)

→ Rotate list

```
int GetLength (struct ListNode * head)
{
    if (head == NULL)
        return 0;
    return 1 + GetLength (head → next);
}
struct ListNode* rotateRight (struct ListNode * head, int k){
    if (head == NULL || k == 0)
        return head;
    int length = GetLength (head);
    if (length == 1)
        return head;
    for (int i = 0; i < k % length ; i++)
    {
        struct ListNode *p = head;
        while (p → next → next != NULL)
        {
            p = p → next;
        }
        struct ListNode *a = (struct ListNode *) malloc (sizeof (struct ListNode
        a → val = p → next → val;
        a → next = head;
        head = a;
        p → next = NULL;
    }
    return head;
}
```

Output :- Input: head = [1, 2, 3, 4, 5], k = 2

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

22-2-24

Week - 09

1) Write a program to traverse a graph using BFS method

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

struct Queue {
    int items[MAX];
    int front;
    int rear;
};
struct Queue * createQueue() {
    struct Queue* queue = (struct Queue*) malloc(sizeof(struct Queue));
    queue -> front = -1;
    queue -> rear = -1;
    return queue;
}
int isEmpty(struct Queue* queue){
    if(queue -> rear == -1)
        return 1;
    else
        return 0;
}
int isFull(struct Queue *queue) {
    if(queue -> rear == MAX-1)
        return 1;
    else
        return 0;
}
void enqueue(struct Queue * queue, int value) {
    if(isFull(queue))
        printf("\n Queue is Full ");
    else {
        if(queue -> front == -1)
            queue -> front = 0;
```

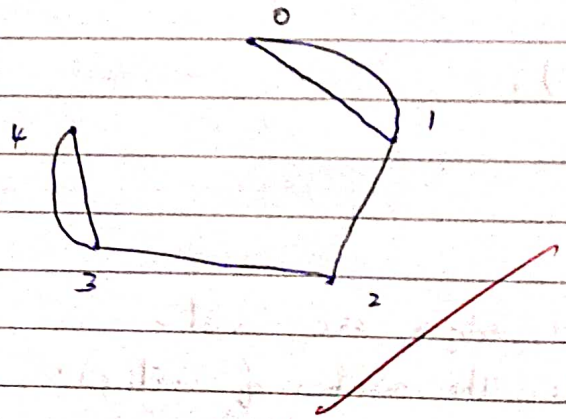
```
        queue->rear++;
        queue->items[queue->rear] = value;
    }
}

int dequeue(struct Queue* queue){
    int item;
    if(isEmpty(queue)){
        printf("Queue is empty");
        item = -1;
    }else{
        item = queue->items[queue->front];
        queue->front++;
        if(queue->front > queue->rear){
            queue->front = queue->rear = -1;
        }
    }
    return item;
}

struct Graph{
    int vertices;
    int** adjMatrix;
};

struct Graph* createGraph(int vertices){
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
    graph->vertices = vertices;
    graph->adjMatrix = (int**)malloc(vertices*sizeof(int*));
    for(int i=0; i<vertices; i++){
        graph->adjMatrix[i] = (int*)malloc(vertices*sizeof(int));
        for(int j=0; j<vertices; j++)
            graph->adjMatrix[i][j] = 0;
    }
    return graph;
}
```

```
void addEdge (struct Graph * graph, int src, int dest){
    graph -> adjMatrix [src][dest] = 1;
    graph -> adjMatrix [dest][src] = 1;
}
void BFS (struct Graph *graph, int startVertex) {
    int visited [MAX] = {0};
    struct Queue * queue = createQueue ();
    visited [startVertex ] = 1;
    enqueue (queue, startVertex);
    printf (" Breadth first search traversal : ");
    while (! isEmpty (queue)) {
        int currentVertex = dequeue (queue);
        printf ("%d ", currentVertex);
        for (int i=0; i<graph -> vertices; i ++) {
            if (graph -> adjMatrix [currentVertex][i ] ==1 && visited [i]==0)
            {
                visited [i] = 1;
                enqueue (queue, i);
            }
        }
    }
    printf ("\n");
}

int main () {
    int vertices, edges, src, dest;
    printf (" Enter the number of vertices : ");
    scanf ("%d", &vertices);
    struct Graph * graph = createGraph (vertices);
    printf (" Enter the number of edges : ");
    scanf ("%d", &edges);
    for (int i=0; i< edges; i++) {
        printf (" Enter edge %d (source destination): ", i+1);
        scanf ("%d %d", &src, &dest);
        addEdge (graph, src, dest);
    }
```

```
    int startVertex;
    printf("Enter the starting vertex for BFS: ");
    scanf("%d", &startVertex);
    BFS(graph, startVertex);
    return 0;
}
```

Output:-

```
Enter the number of vertices: 5
Enter the number of edges: 6
Enter edge 1 (source destination): 0 1
Enter edge 2 (source destination): 0 1
Enter edge 3 (source destination): 1 2
Enter edge 4 (source destination): 2 3
Enter edge 5 (source destination): 3 4
Enter edge 6 (source destination): 4 4
Enter the starting vertex for BFS: 0
Breadth first search Traversal: 0 1 2 3 4
```



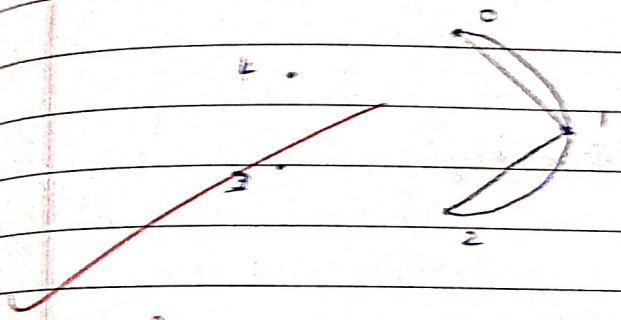
2) Write a program to check whether given graph is connected or not using DFS method.

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

struct Graph {
    int vertices;
    int **adjMatrix;
};
struct Graph * createGraph(int vertices)
{
    struct Graph *graph = (struct Graph *) malloc(sizeof(struct Graph));
    graph -> vertices = vertices;
    graph -> adjMatrix = (int **) malloc(vertices * sizeof(int));
    for(int i=0; i<vertices; i++)
    {
        graph -> adjMatrix[i] = (int *) malloc(vertices * sizeof(int));
        for(int j=0; j<vertices; j++)
            graph -> adjMatrix[i][j] = 0;
    }

    return graph;
}
void addEdge(struct Graph *graph, int src, int dest) {
    graph -> adjMatrix[src][dest] = 1;
    graph -> adjMatrix[dest][src] = 1;
}
void DFS(struct Graph *graph, int startVertex, int visited[])
{

    visited[startVertex] = 1;
    for(int i=0; i<graph -> vertices; i++) {
        if(graph -> adjMatrix[startVertex][i] == 1 && visited[i]==0)
            DFS(graph, i, visited);
    }
}
```

```
int isConnected(struct Graph * graph)
{
    int * visited = (int*) malloc (graph -> vertices * sizeof(int));
    for (int i=0; i< graph -> vertices ; i++)
        visited[i] = 0;
    DFS (graph, 0, visited);
    for(int i = 0; i< graph -> vertices ; i++)
    {
        if (visited[i] == 0)
            return 0;
    }
    return 1;
}

int main()
{
    int vertices, edges, src, dest;
    printf("Enter the number of vertices: ");
    scanf("%d", &vertices);
    struct Graph * graph = createGraph (vertices);
    printf("Enter the number of edges: ");
    scanf("%d", &edges);
    for(int i =0; i<edges ; i++)
    {
        printf("Enter edge %d (source destination): ", i+1);
        scanf("%d%d", &src, &dest);
        addEdge (graph, src, dest);
    }
    if (isConnected(graph))
        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 edge 1 (source destination): 0 1
Enter edge 2 (source destination): 0 1
Enter edge 3 (source destination): 1 2
Enter edge 4 (source destination): 1 2
The graph is not connected



Ex. 1
20/12/24

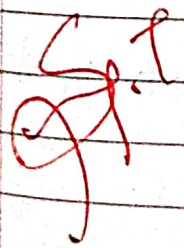
21-2-24

Week-10

WAP for Hashing with Linear Probing

```
#include <stdio.h>
#define SIZE 10

int hashFunction (int key)
{
    return key % SIZE;
}
void insertValue (int hashTable[], int key)
{
    int i=0;
    int hkey = hashFunction(key);
    int index;
    do{
      index = (hkey + i) % SIZE;
      if (hashTable[index] == -1) {
          hashTable[index] = key;
          printf("inserted key %d at index %d\n", key, index);
          return;
      }
      i++;
    } while (i< SIZE);
    printf("unable to insert key %d hash table is full\n", key);
}


int searchValue (int hashTable[], int key)
{
    int i=0;
    int hkey = hashFunction(key);
    int index;
    do{
      index = (hkey + i) % SIZE;
      if (hashTable[index]
      {
```

```
            printf("key %d found at index %d\n", key, index);
            return index;
        }
        i++;
    }while(i < SIZE);
    printf("key %d not found in hash table\n", key);
    return -1;
}

void main()
{
    int hashTable[SIZE];
    for(int i=0; i<SIZE; i++)
    {
        hashTable[i] = -1;
    }
    insertValue(hashTable, 1234);
    insertValue(hashTable, 5678);
    insertValue(hashTable, 9876);
    searchValue(hashTable, 5678);
    searchValue(hashTable, 1111);
    return 0;
}
```



29-2-24

Hackerrank

→ Swap Node [Algo]

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int id;
    int depth;
    struct node *left, *right;
};

void inorder (struct node* tree) {
    if(tree == NULL)
        return;
    inorder (tree → left);
    printf("%d", tree → id );
    inorder ((tree → right));
}
int main (void)
{
    int num, i=0;
    int l, r, max_depth, k;
    struct node *temp = NULL;
    scanf ("%d", &num);
    struct node *tree = (struct node*) calloc (num, sizeof (struct
    tree[0].depth = 1;
    while (i<num) {
        tree[i].id = i+1;
        scanf("%d %d", &l, &r);
        if(l==-1)
            tree[i].left = NULL;
        else {
            tree[i].left = &tree[l-1];
            tree[i].left → depth = tree[i].depth +1;
            max_depth = tree[i].left → depth;
        }
```

```
        if (r == -1)
            tree[i].right = NULL;
        else {
            tree[i].right = &tree[r-1];
            tree[i].right -> depth = tree[i].depth + 1;
            max_depth = tree[i].right -> depth + 2;
        }
        i++;
    }
    scanf("%d", &i);
    while(i--) {
        scanf("%d", &l);
        r = l;
        while (l <= max_depth) {
            for(k=0; k<num; ++k)
            {
                if( tree[k].depth == l)
                {
                    temp = tree[k].left;
                    tree[k].left = tree[k].right;
                    tree[k].right = temp;
                }
            }
            l = l + r;
        }
        inorder(tree);
        printf("\n");
    }
    return 0;
}
```

Sol.1

Output:-

Input:-

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

Output:-

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