INDEX

Name Rushi Hundiwala Class

Roll No. Subject D.S.A. College

Sl. No.	Date	Title	Page No.	Teacher Sign / Remarks
		[illegible]		
WEEK	1	Swapping Using Pointers / D.M.A / Stack implementation		21/12/24
week2	2	infix to postfix evaluation & expression		28/12/24
WEEK	3	Queue & Circular Queue implementation		4/1/24
WEEK	4	Singly Linked List		18/1/24
WEEK	5	Operation on S.L.L and Stack & Que Operations		25/1/24
WEEK	6	implementation of Doubly L.L		1/2/24
WEEK	7	Binary Search Tree		
WEEK	8	Graph Traversal (B.S.T) and (D.F.S)		22/2/24
WEEK	9	Hashing & Linear Probing		29/2/24

WEEK-1

DATE: 21/12/23 PAGE:

Swapping using pointers.

```
#include <stdio.h>

void swap (int *p, int *q){
    int temp;
    temp = *p;
    *p = *q;
    *q = temp;
    printf("The no's you've entered on swapping, are : %d\t%d"
           , *p, *q);
}

void main(){
    int a, b;
    printf("enter two numbers : \n");
    scanf("%d%d", &a, &b);
    swap(&a, &b);
}
```

Dynamic Memory Allocation

(1) Malloc()

DATE: PAGE:

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

void main(){

    int *arr_malloc;
    int n;
    pf("enter a number : \n");
    scanf("%d", &n);
    arr_malloc = (int *) malloc(n * sizeof(int));

    free(arr_malloc);
}
```

(2) Calloc.

```
# include <stdio.h>

void main(){

    int *arr_calloc;
    int n;
    pf("enter a number");
    sf("%d", &n);
    arr_calloc = (int *) calloc(n, sizeof(int));
    fre(arr_calloc);
}
```

DATE: PAGE:

(3) ~~realloc~~ realloc();

```
# include <stdio.h>

int main() {

    int *arr;
    int size1 = 5;

    arr = (int *) malloc( size1 * sizeof(int));

    if ( arr == NULL) {
        pf ("Memory allocation failed \n");
        return 1;
    }

    int size2 = 10;

    int *resizedArr = (int *) realloc( arr,
                      size2 * sizeof (int));

    if ( resizedArray == NULL) {
        pf ("Memory reAllocation failed \n");
        free (arr);
        return 1;
    }

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

21/12/2023

WEEK - 2 21/12/24

stack impl. DATE: PAGE:

* stack implementation

```
#include <stdio.h>
#define max 50;

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

int isFull(){
    return top == max - 1;
}

int isEmpty(){
    return top == -1;
}

int main(){

    int i, a, n;
    pf("enter 1 for push(), 2 for
        3 for display() \n");
    sf("%d", &n);

    if (n == 1){
        pf("enter int input: \n");
        sf("%d", &a);
```

18/11/24

DATE: PAGE:

```
void push(int a) {
        if (isFull()) {
            pf("stack overflow!");
        else {
            stack[++top] = a;
            pf("pushed %d onto the
            stack \n", a);
        }
}
}
else if (n==2) {

void pop() {
        if (isEmpty()) {
            pf("stack underflow");
        else {
            pf("popped %d from the stack")
            , stack[top--]);
        }
}
}

else if (n==3) {

void display() {
        pf("stack : \n");
        for (int i=0; i<=top; ++i)
        {
            pf("%d", stack[i]);
        }
}       pf("\n");
```

WEEK - 3. 4/1/24

DATE: PAGE:

* infix to postfix conversion

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

#define size 100

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

void push(char element){
if (top == size - 1){
    printf("error : stack overflow : \n");
    exit(EXIT_FAILURE);
    }
    stack[++top] = element;
}

char pop(){
    if (top == -1){
        printf("error : stack underflow \n");
        exit(EXIT_FAILURE);
    }
    return stack[top--];
}
```

DATE: PAGE:

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

void infixToPostfix(char infix[]){
    char postfix[size];
    int i,j;
    char symbol;

    push('#');

    for(i=0, j=0; infix[i] != '\0'; i++){
        symbol = infix[i];
        if(isalnum(symbol)){
            postfix[j++] = symbol;
        }
        else if(symbol == '('){
            push(symbol);
        }
```

DATE: PAGE:

```
else if (symbol == ')') {
        while (stack[top] != '#' && stack[top]
!= '(') {
            postfix[j++] = pop();
        }
        pop();
    }
    else {
        while (stack[top] != '#' && precedence
            (symbol) <= precedence (stack[top]))
        {
            postfix[j++] = pop();
        }
        push(symbol);
    }
}
```

18/1/24

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

DATE: PAGE:

```
void main(){
    char infix[size];

    printf("enter infix expression :");
    scanf("%s", infix);

    infixToPostfix(infix);
    return 0;
}
```

Output infix a+b-c/d*e ⇒ ab+cd/e*-

Q.2 Queue Implementation.

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

#define MAX 6

int Q[MAX];
int front = -1, rear = -1;

void insert(){
    if(rear == max-1){
        pf("error : queue is full\n");
    return;
    }

int elemnt;
pf("enter elemnt to be
    added");
sf("%d", &elemnt);
```

DATE: PAGE:

```
if (front == -1) {
        front = 0;
    queue [++rear] = 0;
}

void dulete() {

    if (front == -1 || front > rear) {

        printf ("queue empty \n");

    else
        front++;

    }
}

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

}
```

Singly Linked List

WEEK-4.

DATE: 11/1/24 PAGE:

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

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

void display linked list (struct node *p) {
    printf ("printing linked list : \n");
    while (p != NULL) {
        pf ("%d\n", p -> value);
        p = p -> next;
    }
}

int main () {
    struct node *head = NULL;
    struct node * one = NULL;
    struct node * two = NULL;
    struct node * three = NULL;

    one = malloc (sizeof (struct node));
    two = malloc (sizeof (struct node));
    three = malloc (sizeof (struct node));
```

DATE: PAGE:

```
one -> value = 1;
two -> value = 2;
three -> value = 3;

one -> next = two;
two -> next = three;
three -> next = NULL;

head = one;
display linked list (head);
}
}
```

Output

priting linked list:

1
2
3

11/1/24

WEEK - 4

circular queque

11/1/24

DATE: PAGE:

```
#include <stdio.h>
#include SIZE 100

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 enqueque(int elemnt){
    if(isFull()){
        pf("overflow");
        return;
```

```
if (front == -1) front = 0;
rear = (rear+1) % size;
items[rear] = element;
pf("pushed %d", element);
}

int dequeue(){
    if (isEmpty()) {
        pf("Can't pop, it's empty\n");
        return;
    }
    int element = item[front];
    if (front == rear) {
        front = -1;
        rear = -1;
    }
    else {
        front = (front+1) % size;
    }
    pf("popped %d", element);
    return(element);
}
```

```
int main() {

    enqueue (0);
    enqui (1);
    enique (3);
    enqueue (4);
    enqueue (2);
    dequeue (1);
    display()
}
void display(){

  int i=0

  for( i=front ; i<=rear ; i++){
     pf("%d", &queue[i]);
  };
  return 0;
}
```

WEEK-5 Node Deletion

18/1/24 DATE: PAGE:

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

struct node *head = NULL;

int main(){
        int choice;

        printf(" enter choice /n");
        scanf("%d", &choice);

        switch(choice){

case 1:
                pop();
                break;
case 2:
                end_deleted();
                break;
case 3:
{
        int position;
        pf("enter the position
        from where you wanna
        delete");
        sf("%d", &position);
        delete_at_pos(position); }
```

```
        break;
    case 4 :
        display ():
        break;
    case 5 :
        printf ("existing the program \n");
        break;

    default :
        printf (" invalid choia ");
}

void pop() {
    if ( head == NULL) {
        pf("empty list");
        return;
    }

    if (head -> next == NULL) {
        free(head);

    struct node * ptr = head;
    head = head -> next;
    free (ptr);
    pf(" node : deleted ");

}
```

DATE: PAGE:

```
void end_delete(){

    if(head == NULL){
        printf("List is empty.");
        return}
}

    if (head -> next == NULL){
        free(head);
        head = NULL;
        pf("node deleted from end")
        return;
    }

struct node * ptr = head;
struct node * ptr1 = NULL;

while (ptr -> next != NULL){
        ptr1 = ptr;
        ptr = ptr -> next;
    }

    ptr1 -> next = NULL;
    free(ptr);
    printf("Node deleted from end");
}
```

DATE: PAGE:

```
void delete_at_pos(int position){

    if(head == NULL){
        printf("List is empty");
        return;
    }

    struct node *ptr = head;
    struct node *ptr1 = NULL;

    for(int i = 1; i < position && ptr1 != NULL; i++){
        ptr1 = ptr;
        ptr = ptr->next;
    }

    if(ptr == NULL){
        printf("There are less elements");
        return;
    }

    if(ptr1 == NULL){
        head = ptr->next;
    } else {
        ptr1->next = ptr->next;
    }

    free(ptr);
    pf("Node deleted from position");
```

8/11/24

```
}

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

Output

10 20 30

Node Deleted at pos 2

10 30

week - 5

LeetCode Min stack.

DATE: PAGE:

```
int top = -1;

typedef struct minStack {
        int stack[MAX];
} Minstack;

Minstack * minstackCreate(){
        Minstack * obj = (Minstack *) malloc(
        return obj;
}
void minstackPush(Minstack * obj, int
(Minstack *) malloc(sizeof(Minstack)), val)
{   if (top == MAX - 1) {
        printf("stack overflow \n");
        return; }
    else { obj -> stack[++top] = val; } }
void minStackPop(MinStack * obj) {
    if (top == -1) { pf("underflow"); return; }
    else { top-obj->stack[++top] = val; } };
void
int main() {
    MinStack * stack = minStackCreate();
    minStackPush(stack, 5);
    minStackPush(stack, 10);
    minStackPop(stack);
    free(stack);
    return 0;
}
```

25/1/24. WEEK-6. Reversing, Concatenati

Singly Linked List Sorting, DATE: PAGE:

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

struct node {
    int data;
    struct node * next;
};
// Linked list structure.
struct LinkedList {
    struct node * head;
};

// creating : funct^n for new-node
struct Node* createNode (int data){
    struct node* newNode = (struct Node*)
malloc (sizeof (struct node));
newNode -> data = data;
newNode -> next = NULL;
return newNode; }

// appending node to list funct^n.
void append (struct LinkedList* list,
int data) {
    struct Node* newNode = createN
                           ode(data
if (list -> head == NULL) {
        list -> head = newNode;
        return; }
```

DATE: PAGE:

```
struct node* lastNode = list->head;
while (lastNode->next != NULL) {
    lastNode = lastNode->next;
}
lastNode->next = newNode;
}

// function to display LL.
void display (struct LinkedList* list){
    struct Node* current = list->head;
    while (current != NULL){
        pf("%d", current->data);
        current = current->next;
    }
    pf("\n");
}

void sortList (struct LinkedList* list)
{
    int swapped, temp;
    struct Node* current;
    struct Node* last = NULL;

    do{
        swapped = 0;
        current = list->head;
```

DATE: PAGE:

```
while (current -> next != last) {
    if (current -> next != last) {
    if (current -> data > current -> next
        -> data) {
        // swapping data
        temp = current -> data;
        current -> data = current -> next -> data;
        current -> next -> data = temp;

        swapped = 1;
        }
        current = current -> next;
        }
        last = current; }
        while (swapped);
    }

void reverseList (struct LinkedList * list)
{
    struct Node * prev = NULL, * current
    = list -> head, * nextNode;

    while (current != NULL) {
        nextNode = current -> next;
        current -> next = prev;
        prev = current;
        current = nextNode; }
        list -> head = prev; }
```

DATE: PAGE:

```
void concatenateLists(struct LinkedList *
list1, struct LinkedList* list2){
    if(list1->head == NULL){
        list1->head = list2->head;
    }
    else{
        struct Node* current = list1->head;
        while(current->next != NULL){
            current = current->next;
        }
        current->next = list2->head;
    }
    list2->head = NULL;
}
```

Output

Original Linked List: 3 1 5 2

Sorted Linked List: 1 2 3 5

Reversed Linked List: 2 5 1 3

Concatenated Linked List 5 3 2 1 8 6 7

WEEK-6
25/1/24. stack & queue impl. of. Linked list DATE: PAGE:

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

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

struct stack {
    struct Node *top; }
```

Exit
25/1/24.

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

void push (struct stack *stack, int
  data) {
    struct Node * newNode = createNode (data);
    newNode -> next = stack -> top;
    stack -> top = newNode;
}
```

DATE: PAGE:

```
int pop (struct stack * stack) {
    if ( stack -> top == NULL) {
        printf ("overflow");
        return -1;
    }
    struct Node* temp = stack -> top;
    int poppedData = temp -> data;
    stack -> top = temp -> next;
    free (temp);
    return poppedData;
}

void displayStack ( struct Stack * stack
{
    struct Node * current = stack -> top;
    printf ("Stack: ");
    while ( current != NULL) {
        pf ("%d", current -> data);
        current = current -> next;
    }
    printf ("\n");
}

int main() {
```

DATE: PAGE:

```
struct Stack stack;
stack.top = NULL;

push(&stack, 3);
push(&stack, 1);
push(&stack, 5);

displayStack(&stack);

int poppedData = pop(&stack);

if (poppedData != -1) {
    printf("popped elemnt : %d\n", poppedData);
}

displayStack(&stack);

return 0;

}
```

Output

```
Stack : 5 1 3
Popped Elemnt: 5
Stack : 1 3
```

WEEK-6
queue implementation using linked list.

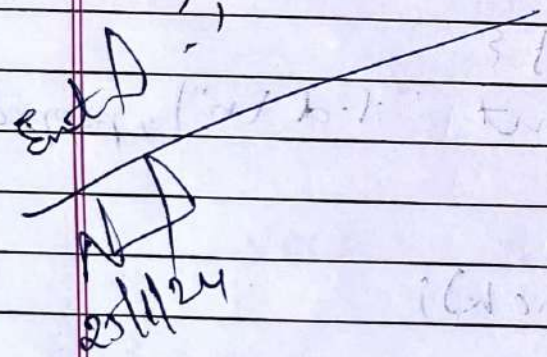
25/1/24.

DATE: PAGE:

```
// queue implementation

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

struct Node *Queue{
        struct Node *front;
        struct Node *rear;
};
```



WEEK-D

Doubly Linked List

1/2/24

DATE:

PAGE:

```
struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};

struct Node* createNode(int data)
{
    struct Node* newNode = malloc(sizeof(struct Node));

    newNode->data = data;
    newNode->link = NULL;
    newNode->next = NULL;
    return newNode;
}

void insertLeftOfNode(struct Node** head, int newValue, int specificValue)
{

    struct Node* newNode = createNode(newVal
    struct Node* current = *head;

    while (current != NULL && current
    ->data != specificValue)
    {
        current = current->next;
    }
```

NLD 1/2/24

```
if (current == NULL) {
        printf ("Node with specific
value not found. \n");
        return ;
    }
    if (current -> prev != NULL) {
        current -> prev -> next = newNode;
        newNode -> prev = current -> prev ;
    }
    else {
        *head = newNode ; }

    newNode -> next = current;
    current -> prev = newNode ;
}

void deleteNodeByValue (struct Node **
head, int deleteValue)

{
    struct Node * current = *head ;

    while (current != NULL && current
-> data != deleteValue)

    {   current = current -> next ; }

    if ( current == NULL) {
        ptf (" Node with specified
            value not found");
        return ; }
```

```
if (current -> prev != NULL) {
    current -> prev -> next = current -> next;
}

if (current -> next != NULL) {
    current -> next -> prev = current
    -> prev;

    free (current);
}

void displayList (struct Node * head)
{
    pf ("%d <-> ", current -> data);

    current = current -> next;
}
    pf ("NULL\n");
}
```

DATE: PAGE:

```
void createList (struct Node ** head)
{
    for (int i = 1; i <= 5; ++i) {
        struct Node * newNode = create
                                Node(i);

    if (* head == NULL) {
        * head = newNode; }
    else {
        (struct Node *) temp = * head;
        while (temp -> next != NULL)
        { temp = temp -> next; }

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

~~int main(){~~
~~struct Node * head = NULL;~~
~~createList (& head);~~

DATE: PAGE:

```
void insert_at_Position(struct Node** head, int newValue, int pos)
{ struct Node * newNode = createNode(newValue);
  struct Node* current = *head;
  if (pos == 1) {
      newNode -> next = current
      if (current != NULL) {
          current -> prev = newNode;
      }
      *head = newNode;
      return;
  }
  int currentPosition = 1;
  while (currentPosition < pos-1 && current != NULL)
  {
  }
int main() {
    struct Node* head = NULL;
    createList(&head);
    insertLeftOfNode(&head, 6, 3);
    insertAtPosition(&head, 10, 2);
    deleteNodeByValue(&head, 2);
    displayList(head);
```

DATE: PAGE:

```
    return 0;
}
```

Output

1 <-> 10 <-> 6 <-> 3 <-> 4 <-> 5 <-> NULL

Leet code - WEEK-7

1/2/24

DATE: PAGE:

```
struct ListNode {
    int val;
    struct ListNode *next;
};

void append( struct ListNode **head, int val)
{
    struct ListNode * new = malloc(sizeof(ListN
    struct ListNode * prev = *head;

    new->val = val;
    new->next = NULL;

    if( *head == NULL) {
            *head = new;
    }
    else {

        while (prev->next != NULL)
    {       prev = prev->next;
            prev->next = new;
    }
}
```

1/2/2024

DATE: PAGE:

```
int length (struct ListNode *head)
{
    struct ListNode * prev = head;
    int len = 0;

    while (prev != NULL) {
        prev = prev -> next;
        len++;
    }
    return len;
}

struct ListNode ** splitListToParts
(struct ListNode * head, int k, int*
returnSize) {

    struct ListNode** heads = (struct
    ListNode **) * k);

    int len = length (head);

    for ( int i = 0; i < k; i++) {

        struct ListNode * nhead = NULL;
        heads [i] = nhead;
    }

    int common = len / k;
    int extra = len % k;
```

```
int iter;
int i=0;

while( prev != NULL) {

    for (j=0 ; j < common + ((extra>0)
                              ? 1: 0); j++)

{   append (&heads[i], prev->val);
        prev = prev->next;
 }
 i++;
 extra--;
}

*returnSize = k;
return heads;
}
```

WEEK-8

15/2/24

Binary Search Tree

DATE: PAGE:

```
struct Node {
    int data;
    struct Node * left;
    struct Node * right;
};

struct Node * createNode (int data) {

    struct Node * newNode = malloc
    (sizeof (struct Node));

    newNode -> data = data;
    newNode -> left = newNode -> right
     = NULL;

    return newNode;
}

struct Node * insert (struct Node * root
, int data) {

  if (root == NULL) {
        root = createNode (data); }
  else if (data <= root -> data) {
        root -> left = insert (root -> left,
    data);
  else {
    root -> right = insert (root -> right
    , data); }
    return root; }
```

DATE: PAGE:

```
void inOrder (struct Node * root) {
    if (root != NULL) {
        inOrder (root -> left)
        pf ("%d", root -> data);
        preOrder (root -> left);
        inOrder (root -> right);
    }
}

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

void display (struct Node * root) {
    printf ("in-order traversal : ");
    inOrder (root);
    pf ("\n preOrder traversal: ");
    preOrder (root)
    pf ("\ post - order ");
    postOrder (root);
    pf ("\n");
}
```

DATE: PAGE:

```
int main () {

    struct Node *root = NULL

    root = insert (root, 10);
    root = insert (root, 5);
    root = insert (root, 15);
    "  = "  ( " , 7);
    "  = "  ( " , 3);
    "  = "  ( " , 12);

    display (root);

    return 0;
}

void preOrder (struct Node *root)
{
    if (root != NULL) {
        printf ("%d" -> root -> data);
        inOrder (root -> left);
        preOrder (root -> right);
    }
}
```

Output: 10, 5, 15, 7, 3, 12

inOrder Traversal : 3 5 7 10 12 15
preOrder Traversal 10 5 3 7 15 12
post Order Traversal - 3 7 5 12 15 10

Week - 8 15/2/24.

LeetCode DATE: PAGE:

```
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 *p= head;
    struct ListNode *a= malloc(sizeof
                        (struct ListNode));
    a->val = p->next->val;
    a->next = head;
    head = a;
    p->next = NULL; }
```

22/2/24

WEEK-9. BFS / DFS

DATE: PAGE:

```
void addEdge (int start, int end) {
    adjacencyMat [start][end] = 1;
    adjacencyMat [end][start] = 1;
}

int adjacencyMat [MAX_VERTICES][MAX_VERTICES];

void ~~bfsTraversal~~ bfsTravorsal (int numVertices
    , int startVertex) {

    int queque [MAX_VERTICES];
    int front = 0, rear = 0;
    int visited [MAX_VERTICES] = {0};
    int i = 0;

    visited [startVertex] = 1;
    queque [rear++] = startVertex;

    while (front != rear) {

        int currentVertex = queque [front++];
        pf ("%d", currentVertex);
        for (i = 0 ; i < numVertices ; i++) {
            if (adjacencyMatrix [currentVertex][i]
            && !visited [i]) {

                visited [i] = 1;
                queque [rear++] = i;
            }
        }
    }
```

DATE: PAGE:

```
int isConnected(int numVertices){
    int visited[MAX_VERTICES] = {0};
    int i;

    void dfsUtil(int vertex){
        visited[vertex] = 1;
        for(i=0; i<numVertices; i++){
            if(adjacencyMatrix[vertex][i] &&
               !visited[i]){
                dfsUtil(i);
            }
        }
    }
    dfsUtil(0);

    for(i=0; i<numVertices; i++){
        if(!visited[i]){
            return 0;
        }
    }
    return 1;
}
```

DATE: PAGE:

```
int main(){

    int numVertices, startVertex;
    int i,j;

    printf("enter the no. of vertices:");
    scanf("%d", &numVertices);

    printf("enter the adjacency
                matrix:\n");
    scanf("%d", &numVertices);

    printf
    for(i=0; i<numVertices; i++){
        for(j=0; j<numVertices; j++){
            scanf("%d", &adjacencyMat[i][j]);
        }
    }

    printf("enter the starting vertex for
                BFS traversal:");
    scanf("%d", &startVertex);

    return 0;
}
```

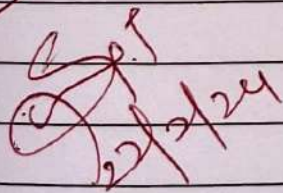
DATE: PAGE:

Output

Enter the number of vertices : 2
Enter the adjacency matrix :
1
2
3
4
Enter the starting vertex for BFS traversal : 2

BFS traversal starting from vertex 2 :

The graph is connected



22/2/24

LeetCode : Swap Nodes [Algo]

DATE: PAGE:

```
#include <stdio.h>
#include <stdlib.h>
#define S(a) scanf("%d", &a)
#define SS(a,b) scanf("%d %d", &a, &b)

struct node{
    int data;
    int left;
    int right;
};

void swapNodes(struct node *arr
, int i, int level, int k) {

    if (i == -1) {
        return;
    }

else {
    if (level == k) {
        swap(&arr[i].left, &arr[i].right
        return;
    }
    else if (level < k) {
        int left = arr[i].left;
        int right = arr[i].right;
        level++;
        swapNodes(arr, left, level, k);
        swapNodes(arr, right, level, k
        return;}
```

DATE: PAGE:

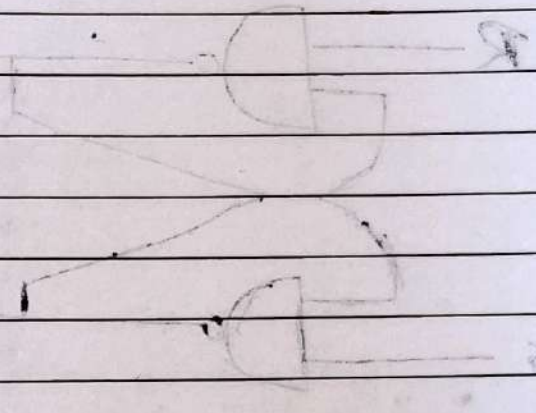
else {

return;

}

}

Sri
22/2/24.



WEEK-10

29/2/24

DATE: PAGE:

Hash Table & ~~Prob~~ Linear Probing

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

int hash_table[TABLE_SIZE];

void initializeHashTable(){
    int i;
    for(i=0; i<TABLE_SIZE; i++){
        hash_table[i] = -1;
    }
}

int hash(int key){
    return key % TABLE_SIZE;
}

void insert(int key){
    int hkey = hash(key);
    int index = hkey;
    int i = 0;
```

DATE: PAGE:

```
while (hash_table[index] != -1) {
    i++;
    index = (hkey + i) % TABLE_SIZE;

    if (hash_table[index] == -1 || i = TABLE_SIZE)
    { printf ("key %d not found\n", key);
      return;
    }
}
printf ("key %d found at index %d\n", key, index);

void displayHashTable() {
    int i;
    printf ("Hash Table:\n");
    for (i = 0; i < TABLE_SIZE; i++) {
        pf("%d :", i);
    if( hash-table[i] != -1) {
        printf ("%d", hash-table[i]); }
    printf ("\n");
    }
}

int main() {
    int i, a, b, c, d, e;
    int s1, s2, s3;
    initializeHashTable();
```

WEEK-10

DATE: PAGE:

```
int main (){

    int   a,b,c,d ;
    int   s1,s2;
    initializeHashTable();
    pf(" enter employee number ");
    scanf("%d",&a);
    pf(" enter emp.no. ");
    sf("%d",&b);
    pf("enter emp no");
    sf("%d",&c);
    pf("enter emp no");
    sf("%d",&d);
    insert(a);insert(b);insert(c);insert(d);
    displayHashTable()
    printf(" enter emp. no to search");
    sf("%d",&s1);
    search(s1)
}
```

Output

```
enter employee number : 12
enter employee number : 13
enter employee number : 24
enter employee number : 25
enter emp. no to search : 24

employee number 24 found at
        index 4
```