INDEX

NAME: Prajwal.P STD.: SEC.: ROLL NO.: SUB.:

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1	21/12/23	Stack Application	10	21/12/23
2	28/12/23	Queues, Circular Queeue, Postfix and Infix	10	11/1/24.
3	11/1/24	Linked List -> Insertion	10	
4	18/1/24	Linked List -> Deletion	10	13/1/24
5	18/1/24	Minstack [Leetcode]		
6	18/1/24	Reverse List [Leetcode]	10	28/1/24
7	25-1-24	Linked List Operations	10	
8	1-2-24	Doubly Linked List	10	1/2/24
9	15/2/24	Binary Search Tree	10	22/2
10	22/2/24	BFS_DFS	10	
11	29/2/24	Hashing & Linear Probing.	10	
			10	

classmate
Date
Page

3/12/23

Week 0 - Practice Programs

1) Enter
1 to make a new account
2 to withdraw
3 to deposit
4 to check balance
5 to exit
1
Enter account name
Prajwal.P
Enter age
19
3
Enter amount to deposit
5000
2
Enter amount to withdraw
1500
4
Balance is 3500
5

2) Enter element to search
3

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

3 is present

3) Enter string
My name is Prajwal.P
Enter substring
My
Substring is present

4) Enter elements
12 22 33 44 58 69 77 84 91 22
Enter element to search
22
Last occurance is at 10

5) Enter elements
12 22 33 44 58 69 77 84 91 101
Enter element to search
22
Element 22 is present at 2

6) Enter elements

12 22 33 44 58 69 77 84 91 102

Enter element to search

22

Element 22 is at 2

7) Enter number of elements 5

10 20 30 40 50

Biggest is 50

Smallest is 10

8) Given array ["Hello", "my", "Name", "is", "Prajwal"]

After sorting Lexicographically

Hello

Name

Prajwal

is

my.

21/12/23, Week 1

(1) Swap using pointers

```
#include<stdio.h>
void swap(int *a, int *b);
void main() { int a,b;
    printf("Enter a and b:\n");
    scanf("%d %d",&a,&b);
    swap(a,b);
    printf("After swap a=%d, b=%d",a,b);
}
void swap(int *a, int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

Output

Enter a and b:

5 10

After swapping a: 10 b: 5

(2) Dynamic Memory Allocation

```
#include<stdio.h>
int main(){ int *ptr,n;
printf("Enter number of element : ");
scanf("%d",&n);
ptr = (int*) malloc (n*sizeof(int));
printf("Memory allocated using malloc.\n");
printf("%d Element of array\n");
for(int i=0 ; i<n ; i++) {
    scanf("%d", &ptr[i]);
    }
printf("Elements of array :");
for(int i=0; i<n; i++) {
    printf("%d ",ptr[i]);}}

free(ptr);
ptr = (int*) calloc(n, sizeof(int));
printf("Memory succesfully assigned using calloc ");
for(int i=0; i<n; i++) {
    scanf("%d", &ptr[i]);

printf("Enter new size for realloc \n");
scanf("%d",&n1);
ptr = (int*) realloc(ptr, n1*sizeof(int));
printf("Enter Realloc successful");
```

```
printf("Enter more variables");
for(int i=n; i<n1; i++){
    scanf("%d", &ptr[i]);
}

free(ptr);
printf("Memory freed in");
}
```

Output:-

```
Enter number of elements: 3
Memory allocation by malloc successful
Enter elements of array
1
2
3
The elements of array are 1, 2, 3
Memory freed
Memory successfully allocated using calloc
Enter elements of array
1
2
3
```

Elements of array are 1, 2, 3
Enter new size of realloc : 5
Enter more variables
4
5
Elements of array 1, 2, 3, 4, 5
Memory freed.

(3) Stack application

```
#include<stdio.h>
#include<stdlib.h>
#define max 10
void push(int a[]);
int pop();
void display(int a[]);
int num[max], top = -1;

void main() {
    int n,a;
    printf("Enter \n 1 to push into stack \n 2 to pop stack \n 3 to di
    scanf("%d", &n);
    while(n!=3){
        switch(n){
            case 1: printf("Enter element to push into stack");
                    scanf("%d", &a);
                    push(a)
                    break;
```

```
case 2:a=pop();
        printf("Popped element : %d", a);
        break;
default: printf("Enter correct number");
}
printf("Enter \n 1 to push \n 2 to pop \n 3 to display");
scanf("%d", &n);
}
if(n==3){
    display(num);
}
}
void push (int a){
    if(top==max-1)
      printf("Stack overflow");
    else{
        ++top;
        num[top]=a;
    }
}
int pop()
    if(top==-1)
        printf("Stack underflow");
    else{ return num[top];
        --top;
    }
}
```

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

Output :-

Enter
1 to push into stack
2 to pop stack
3 to display stack
1

Enter element to push
1

1

Enter element to push
2

1

Enter element to push
3

1

Enter elements to push
5

2

Popped element : 5
3
Elements of stack : 1 2 3

21/12/23

Lab 28/12/23

→ Infix to Postfix

```
#include<stdio.h>
#include<ctype.h>
#define max 20
void push(char a);
char pop();
char stack[max], top=-1;
int pre(char a);

void main(){
    char infix[max],a;
    char post[max];
    printf("Enter infix");
    scanf("%s",infix);
    int j=0;
    for(int i=0; i<strlen(infix); i++){
        if((infix[i]=='+' || infix[i]=='-' || infix[i]=='*' ||
            if(pre(infix[i])>pre(stack[top])){
                push(infix[i]);
            else if{
                while(1){
                a=pop();
                if(a=='('){
                push(a);
                break;
```

```
        postf[j] = a;
        j++;
    }
    push(infix[i]);
    }
  }
}
while(top != -1){
    char y = pop();
    if(y == '('){
        break;
    }
    post[j] = y;
    j++;
}
post[j] = '\0';
printf("%s", post);
}

void push(char a){
    if(top == -1){
        printf("Underflow");
        exit(0);
    }
}
```

```
char pop(){
    if (top == -1){
        printf("Underflow");
        exit(0);
    }
    else {
        return stack[top--];
    }
}
int pre(char a){
    if(a=='^'){
        return 3;
    else if (a=='*' || a=='/'){
        return 2;
    else if(a=='+' || a=='-')
        return 1;
    else
        return 0;
    }
```

Output :-

Enter infix :- 2+7*(5%4)-6

Postfix :- 2754%*+6-

→ Postfix evaluation

```
#include <ctype.h>
#include<stdio.h>
#include<string.h>
#define max 20

void push(float a);
int pop();
int stack[max], top=-1;
float compute(int a, int b, char c);

void main() {
    char postfix[max];
    int a,b;
    char c,d;
    printf("Enter postfix expression:");
    scanf("%s", postfix);
    int j=0;
    for(int i=0; i<strlen(postfix); i++){
        if((postfix=="+" || "*" || postfix=="/" || "-"))
            a=pop();
            b=pop();
            c=compute(a,b,postfix[i]);
            push(c);
    }
```

```
    printf("%d", pop());
}
void push(float a){
    if(top > max-1){
        printf("Overflow");
    }
    else{ ++top; stack[top] = a;
    }
}
int pop(){
    if(top == -1){
        printf("Underflow");}
    else { return [stack[top-1]];}
}
}
float compute(int a, int b, char c){
    if(c == '+')
        return a+b;
    else if(c == '*')
        return a*b;
    else if(c == '-')
        return a-b;
    else if(c == '/')
        return a/b;
}
```

Output :-
Enter postfix :- 345*6/9%+
Result :- 6

→ Queue Implementation

```
#include<stdio.h>
#include<stdlib.h>
#define max 20
int queue[max];
int dequeue();
void enqueue(int a);
void display();

void main(){
    int n,m;
    scanf("%d",&n);
    while(n!=3){
        switch(n){
        case 1: printf("Enter element");
                scanf("%d",&m);
                enqueue(m);
                break;
        case 2: m=dequeue();
                printf("Removed element %d",m);
                break;
```

```
case default: print("Invalid input");
    }
    scanf("%d",&n);
    printf("\n");
}}
void enqueue(int a){
    if(rear==max-1){ printf("Overflow");}
    rear+=1;
    queue[rear]=a;
  }
int dequeue(){
    if(rear==-1 || front==rear){ printf("Underflow");
      exit(0);}
    else{ front+=1; return queue[front];}}
  }
}
void display(){
    printf("Queue:\n");
    for(int i=front+1; i<=rear; i++){
        printf("%d\t",queue[i]);
    }
}
```

Output :-

1. Insert element
2. Delete element
3. Exit

1.

Enter element : 23

1. Insert element
2. Delete element
3. Exit

1

Enter element : 54

1. Insert element
2. Delete element
3. Exit

2

Deleted element : 23

1. Insert element
2. Delete element
3. Exit

1.

Enter element : 65

Overflow.

→ Circular Queue

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define max 6;

int cq[max];
int front = -1, rear = -1;

bool is_full(){
    return (rear+1)%max == front; }

bool is_empty(){
    return front==-1 && rear==-1; }

void insert(int item){
    if(is_full()){
        printf("Overflow");
        return;
    }
    if(is_empty()){
        front = rear = 0; }
    else{ rear = (rear+1)%max; }
    cq[rear] = item;
    printf("Enqueued %d", item);
}
```

```
int dequeue(){
    if(is_empty()){
        printf("Underflow");
        return -1;
    }
    int deletedItem = cq[front];
    if(front == rear){
        front = rear = -1;
    }else{ front = (front+1)%max;
    }
    printf("Dequeued: %d ", deleteditem);
    return deleteditem;
}

int main(){
    int n, ele;
    do{
        printf("\n1. Insert\n2. Delete\n3. Exit\n");
        scanf("%d",&n);
        switch(n){
        case 1:
            printf("Enter element: ");
            scanf("%d",&ele);
            insert(ele);
            break;
```

```
case 2:
    int deletedItem = deque();
    if(deletedItem != -1){
        printf("Element removed %d", deletedItem);
    }
    break;
case 3:
    printf("Thank");
}
}while(ch!=3);
return 0;
}
```

Output :-

1. Insert
2. Delete
3. Exit

1

Enter element : 23

1. Insert
2. Delete
3. Exit

2

Dequed element : 23

11-01-24

—> Linked List-

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

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

Node* head = NULL;

void push();
void append();
void insert();
void display();

int main() {
    int choice;
    while(1) {
        printf("1. Insert at beginning\n");
        printf("2. Insert at end\n");
        printf("3. Insert at a position\n");
        printf("4. Display\n");
        printf("5. Exit\n");
        scanf("%d", &choice);
```

```
switch(choice){
    case 1: push(); break;
    case 2: append(); break;
    case 3: insert(); break;
    cae 4: display(); breaks
    default: printf("Exit"); return 0;
  }
 }
}

void push(){
    Node* temp = (Node*) malloc (sizeof (Node));
    int new_data;
    printf("Enter data in node");
    scanf("%d", &new_data);
    temp->data = new_data;
    temp->next = head;
    head = temp;
}

void append(){
    Node* temp = (Node*) malloc (sizeof (Node));
    int new_data;
    printf("Enter data in node");
    temp->data = new_data;
    temp->next = NULL;
```

```
Node* temp1 = head;
while (temp1 -> next != NULL) {
    temp1 = temp1 -> next;
}
temp1 -> next = temp;
}

void insert() {
    Node* temp = (Node*) malloc (sizeof (Node));
    int new_data, pos;
    printf("Enter data in node ");
    scanf("%d", &new_data);
    printf("Enter position ");
    scanf("%d", &pos);
    if(pos == 0){
        temp -> next = head;
        head = temp;
        return;
    }
    Node* temp1 = head;
    while(pos--){
        temp1 = temp1 -> next;
    }
```

```
Node* temp2 = temp1->next
 temp->next = temp2;
 temp1->next = temp;
}

void display() {
    Node* temp1 = head;
    while (temp1 != NULL) {
        printf("%d" ->", temp1->data);
        temp1 = temp1->next;
    }
    printf("NULL\n");
}
```

Output

1. Insert at Beginning
2. Insert at End
3. Insert at position
4. Display
5. Exit

Enter choice: 1

Enter data in node: 10

1. Insert at Beginning
2. Insert at End
3. Insert at position
4. Display

5. Exit
Enter choice : 2
Enter data in node : 20

1. Enter at Beginning
2. Enter at End
3. Enter at position
4. ~~Enter~~ Display
5. Exit
Enter choice : 3
Enter data in node : 15
Enter position : 1
$

1. Enter at Beginning
2. Enter at End
3. Enter at position
4. Display
5. Exit
Enter choice : 4
10 → 20 → 15 → NULL

18-01-24

Linked List (Delete)

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

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

Node* head = NULL;

void display();
void create()
void idelete();
void ldelete();
void fdelete();

int main() {
      create();
      int choice;
      display();
      while(1) {
            printf("1. Delete first element\n");
            printf("2. Delete last element\n");
            printf("3. Delete specific position\n");
```

```
printf("4. Display\n");
printf("5. Exit\n");
scanf("%d", &choice);
switch (choice) {
    case 1: fdelete();
            break;
    case 2: ldelete();
            break;
    case 3: idelete();
            break;
    case 4: display();
            break;
    default: printf("Exiting");
             return 0;
  }
 }
}

void fdelete() {
    if (head == NULL)
        printf("Empty List");
    Node *ptr = head;
    Node *ptr1;
    while (ptr -> next != NULL) {
            ptr1 = ptr;
```

```
        ptr = ptr -> next;
    }
    ptr1 -> next = NULL;
    free(ptr);
}

void fdelete() {
    if (head == NULL)
        printf("Empty");
    Node *ptr = head;
    head = head -> next;
    free(ptr);
}

void idelete() {
    printf("Enter position");
    int pos;
    scanf("%d", pos);
    if (head == NULL)
        printf("Empty list");
    Node *ptr1 = head;
    Node *ptr = head;
    pos--;
    while(pos--) {
        ptr1 = ptr;
        ptr = ptr -> next;
```

```
    ptr1 -> next = ptr -> next;
    free (ptr);
}

void display() {
    Node * temp1 = head;
    while (temp1 -> next != NULL) {
        printf ("%d ->", temp1 -> data);
        temp1 = temp1 -> next;
    }
    printf("NULL \n");
}

void create(){
    int A[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    struct Node *t, *last;
    head = (struct Node*) malloc (size( struct Node));
    head -> data = A[0]
    head -> next = NULL;
    last = head;
    for ( int i = 1; i < 10; i++) {
        t = (struct Node*) malloc (sizeof (struct Node));
        t -> data = A[i];
        t -> next = NULL;
        last -> next = t
        last = t;
    }
}
```

o/p

0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> NULL

1. Delete 1st Element
2. Delete 2nd element
3. Delete specific position
4. Exit

Enter choice: 1

1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> NULL

1. Delete 1st Element
2. Delete 2nd element
3. Delete specific element
4. Exit

Enter choice: 1

1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> NULL

1. Delete 1st element
2. Delete 2nd element
3. Delete specific element
4. Exit

Enter choice: 3

Enter position: 4

1 -> 2 -> 3 -> 5 -> 6 -> 7 -> 8 -> NULL

Exiting program

18/1/24

→ Minstack

```
#include <stdio.h>
#include <stdlib.h>
#define max 1000

typedef struct {
    int top;
    int st[max];
    int min[max];
} Minstack;

Minstack * minstackCreate() {
    Minstack* stack = (minstack*) malloc(sizeof(minstack));
    stack->top = -1;
    return stack;
}

void minStack(Minstack* obj, int val) {
    if(obj->top == max-1) {
        printf("Full");
        return;
    }
    obj->st[++obj->top] = val;
```

```
if(obj->top > 0) {
    if(obj->min[obj->top-1] < val)
        obj->min[obj->top] = obj->min[obj->top-1];
    else
        obj->min[obj->top] = val;
}

void minPop(Minstack * obj){
    if (obj->top == -1){
        printf("Empty");
    }
    else
        obj->top = -1;
    }
}

int minTop(Minstack * obj){
    if( obj->top == -1) {
        printf("Empty");
        return -1;
    }
    return obj->st[obj->top];
}
```

```
int minGetMin (MinStack * obj){
    if(obj->tob == -1){
    printf("Empty");
    return -1;
    }
    return obj->min[obj->tob];
}

void frem (MinStack* obj){
    free(obj);
}
```

Output:-

Min: 1

Top: 1

Min: 1

→ Reversing Linked list

```
struct ListNode* reverseBetween(struct ListNode* head, left
    struct ListNode* ptr1 = head;                    , right)
    int temp = left - 1;
    while(temp--){
        ptr = ptr1->next;
    }
    int count = right - left + 1;
    int* a = (int*)malloc(count * sizeof(int));
    for(int i=0; i<count; i++){
        a[i] = ptr1->val;
        ptr1 = ptr1->next;
    }
    struct ListNode* ptr = head;
    left--;
    ~~while(left--){~~
        ~~ptr = ptr->next;~~
    while(left--){
        ptr = ptr->next;
    }
    for(int i=count-1; i>-1; i--){
        ptr->val = a[i];
        ptr = ptr->next;
    }
    return head;
}
```

25-01-24

→ List operations

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct node * next;
};
struct node *head, *headB;
void createA() {
    int A = {2,1,3,5,4,8,6,7,0,9};
    printf("Elements of Linked List A : ");
    for (int i = 0; i < 10; i++) {
        printf("%d -> ", A[i]);
    }
    printf("NULL");
    struct node *t, *last;
    head = (struct node *) malloc (sizeof (struct node));
    head -> data = A[0];
    head -> next = NULL;
    last = head;
    for (int i = 0; i < 10; i++) {
        t = (struct node *) malloc (sizeof (struct node));
        t -> data = A[i];
        t -> next = NULL; last -> next = t
        last = t;
    }
```

```
void createB() {
    int B[] = {11, 13, 12, 10, 14};
    printf("Elements of Linked List B : ");
    for (int i = 0; i < 5; i++) {
        printf("%d -> ", A[i]);
    }
    printf("NULL");
    struct node *t, *last;
    headB = (struct node *) malloc (sizeof (struct node));
    headB->data = A[0];
    headB->next = NULL;
    last = head;
    for (int i = 1; i < 5; i++) {
        t = (struct node *) malloc (sizeof (struct node));
        t->data = A[i];
        t->next = NULL;
        last->next = t;
        last = t;
    }
}

struct node * sort (struct node *head) {
    struct node *ptr1, *ptr2;
    ptr1 = head;
```

```
int temp, count = 1;
while (ptr1 -> next != NULL) {
        count++;
        ptr1 = ptr1 -> next
}
while (--count) {
  ptr1 = head;
  while (ptr1 -> next != NULL) {
      ptr2 = ptr1;
      ptr1 = ptr1 -> next;
      if (ptr1 -> data < ptr2 -> data) {
          temp = ptr2 -> data;
          ptr2 -> data = ptr1 -> data;
          ptr1 -> data = temp;
      }
  }
} return head; }
```

```
struct node* concat (struct node *head, struct node *head
    struct node *ptr = head;
    while (ptr -> next != NULL) {
        ptr = ptr -> next;
    }
    ptr -> next = headB;
    return head; }
```

```
struct node *reverse (struct node *head){
    struct node *pre, *cur, *after;
    pre = NULL;
    after = cur = head;
    while (after != NULL) {
        after = after->next;
        cur->next = pre
        pre = cur
        cur = after;
    }
    head = pre
    return head;
}

void main() {
    createA();
    createB();
    head = sort(head);
    struct node* p = head;
    printf("Sorted List A: ");
    while(p != NULL) {
        printf("%d -> ", p->data);
        p = p->next;
    } printf("NULL");
    headB = sort(sort(headB));
    p = headB;
```

```
printf("\n Sorted List B: ");
while (p1 = NULL) {
    printf("%d -> ", p->data);
    p = p->next;
} printf("NULL");
p = concat(head, headB);
printf("Concatenation of sorted A and B: ");
while(p1 = NULL){
    printf("%d -> ", p->data);
    p = p->next;
} printf("NULL");
p = reverse(head);
printf("Reverse of concat of A and B: ");
while(p1 = NULL){
    printf("%d -> ", p->data);
    p = p->next;
} printf("NULL");
}
```

Output:-

Elements of Linked List A:

2 → 1 → 3 → 5 → 4 → 8 → 6 → 7 → 0 → 9 → NULL

Elements of Linked List B:

11 → 13 → 12 → 10 → 14 → NULL

Sorted Linked List A:

0 → 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → NULL

Sorted Linked List B:

10 → 11 → 12 → 13 → 14 → NULL

Concatenation of sorted A and B:

0 → 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10 → 11 → 12 → 13 → 14 → NULL

Reverse of A concat B:

14 → 13 → 12 → 11 → 10 → 9 → 8 → 7 → 6 → 5 → 4 → 3 → 2 → 1 → 0 → NULL

28/1/24

Queue - Linked List

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

Node *head = NULL

void enqueue() {
    Node *ptr = (Node*) malloc (sizeof(Node));
    int num;
    num = scanf();
    ptr->val = num;
    if (head == NULL) {
        head = ptr;
        ptr->next = NULL;
    }
    else {
        Node *ptr2 = head;
        while (ptr->next != NULL) {
            ptr2 = ptr2->ptr->next
        }
        ptr2->next = ptr; }}
```

```
void display() {
    Node *ptr = head;
    while(ptr != NULL) {
        printf("%d ", ptr->val);
            ptr = ptr->next;
    }
    printf("\n");
}

void dequeue() {
    Node *ptr = head;
    int num = ptr->val;
    head = head->next;
    free(ptr);
    printf("Dequeued element: %d \n", num);
}

int main() {
  int choice;
  printf("Enter choice: ");
  while(1) {
      scanf("%d", &choice);
      switch(choice){
        case 1: enqueue(); break;
        case 2: dequeue(); break;
        case 3: display(); break;
```

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

Output :-

Enter choice : 1
Enter element to enqueue : 10
Enter choice : 1
Enter element to enqueue : 20
Enter choice : 2
Dequeued element : 10
Enter choice : 3
20
Enter choice : 4
Exiting program.

Stack - Linked List

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

struct Node *top =NULL;

void pop(){
    if(top==NULL){
        printf("Empty stack");
    }
    struct Node *ptr = top
    top = top->next;
    int num = ptr->val;
    free(ptr);
    printf("Popped element : %d \n", num);
}
void push() {
    struct Node *ptr = (struct Node *)malloc(sizeof(struct Node));
    int new_data;
    scanf("%d", &new_data);
    temp->val = data;
    temp->next = top;
```

```
    top = temp;
}

void display() {
    struct Node *temp1 = top;
    while (temp1 != NULL) {
        printf("%d\n", temp->val);
        temp1 = temp1->next;
    }
}

void main(){
    int choice;
    printf("Enter choice:");
    while(1) {
        scanf("%d", &choice);
        switch(choice){
            case 1: push(); break;
            case 2: pop(); break;
            case 3: display(); break;
            case 4: printf("Exit");
                    return 0;
        }
    }
}
```

Output

Enter choice: 1

Enter element to push: 10

Enter choice: 1

Enter element to push: 20

Enter choice: 2

Popped element: 20

Enter choice: 3

10

Enter choice: 4

Exiting the program.

1/2/24

→ Week 6 Doubly Linked list

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

Node *head = NULL;

void insert(){
    int num, pos;
    printf("Enter value : ");
    scanf("%d", &num);
    printf("Enter node to insert left of : ");
    scanf("%d", &pos);

    Node *ptr = (Node*) malloc (sizeof (Node));
    ptr -> val = num;

    if(pos == 0){
        ptr -> next = head;
        ptr -> prev = NULL;
        if (head != NULL){
            head -> prev = ptr;
        }
```

```
    head = ptr;
}
Node *ptr1 = head;
if(pos!=0){
    for(int i=0; i<pos; i++){
        ptr1 = ptr1->next;
    }
    ptr->next = ptr1;
    ptr->prev = ptr1->prev;
    ptr1->prev->next = ptr;
    ptr1->prev = ptr;
  }
}

void delete(){
    printf("Enter value to delet: ");
    int loc=-1, len=1, val;
    scanf("%d", &val);
    Node *ptr = head, *ptr2;
    while(ptr->next!=NULL){
        len++;
        ptr = ptr->next;
    }
    ptr = head;
```

```
if (loc == -1) {
    printf("Delete element not in list");
    return;
}
if (loc == 0) {
    printf("Deleted element : %d \n", ptr->val);
    ptr = head;
    head = head->next;
    free(ptr);
    return;
}
if (loc == len) {
    ptr = head;
    while (ptr->next != NULL) {
        ptr = ptr->next;
    }
    ptr->prev->next = NULL;
    free(ptr);
    return;
}
ptr = head;
for (int i = 0; i < loc; i++) {
    ptr2 = ptr;
    ptr = ptr->next;
}
```

```
    ptr2 -> next = ptr -> next;
    ptr -> next -> prev = ptr2;
    free(ptr);
}

void display() {
    Node *ptr = head;
    while(ptr != NULL) {
        printf("%d <-> ", ptr -> val);
        ptr = ptr -> next;
    }
    printf("NULL");
}

void main() {
    int choice;
    printf("1. To insert val to left");
    printf("2. To delete from any value");
    printf("3. To display list");
    while(1) {
        switch(choice) {
            case 1: insert(); break;
            case 2: delete(); break;
            case 3: display(); break;
            case 4:
            default: printf("Exiting");
                     return;
```

}
}
}

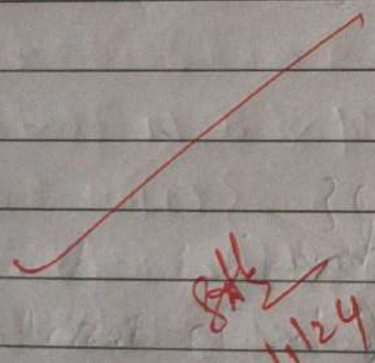
o/p

1. To insert into left of List
2. To delete from
3. To display list

(0) Enter choice : 1
Enter node to insert of : 0
Enter val : 10

(0) Enter ~~node~~ choice : 1
Enter val : 20
Enter node : 0

(0) Enter choice : 1
Enter val : 30
Enter node : 1



(1) Enter choice : 3
20 <-> 30 <-> 10 <-> NULL

(1) Enter choice : 2
Enter ~~node~~ val : 30
Deleted element : 30

(1) Enter choice : 3
30 <-> 10 <-> NULL

Binary Search Tree

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

Struct Tree {
    int val;
    struct Tree *left, *right;
};

Struct Tree *root = NULL;

struct Tree* newNode(int a) {
    Struct Tree *ptr = (struct Tree*) malloc (sizeof(struct Tree));
    ptr->val=a;
    ptr->left = ptr->right = NULL;
    return ptr;
}

struct Tree * insert(struct Tree *ptr, int a){
    if(ptr==NULL){
        return newNode(a);
    }
    else if (ptr->val<a){
        ptr->right = insert(ptr->right, a);
    }
```

```
        else if (ptr->val > a) {
            ptr->left = insert(ptr->left, a);
        }
    }

Pre
void Order(struct Node *ptr) {
    if(ptr != NULL) {
        printf("%d ", ptr->val);
        PreOder(ptr->left);
        PreOrder(ptr->right);
    }
}

void inOrder(struct Node *ptr) {
    if(ptr != NULL) {
        inOrder(ptr->left);
        printf("%d ", ptr->val);
        inOrder(ptr->right);
    }
}

Void PostOrder(struct Tree *ptr) {
    if(ptr != NULL) PostOrder(ptr->left);
        Postorder(ptr->right);
        printf("%d", ptr->val);
}
```

```
void Main(){
    int arr[]{ 8, 5, 6, 1, 2, 3, 7, 4, 9, 0};
    printf("Array to sort : %d", arr[0]);
    root = insert(root, arr[0]);

    for(int i = 1; i<10; i++){
        printf("%d ", arr[i]);
        insert(root, arr[i]);
    }
printf("\n InOrder : ");
inOrder(root);
printf("\n PreOrder : ");
PreOrder(root);
printf("\n Post Order : ");
PostOrder(root);
}
```

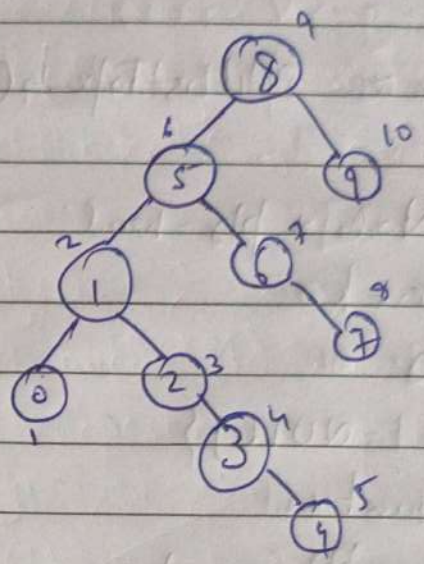
O/P

Array to sort : 8 5 6 1 2 3 7 4 9 0
PreOrder : 8 5 1 0 2 3 4 6 7 9
PostOrder : 0 4 3 2 1 7 6 5 9 8
InOrder : 0 1 2 3 4 5 6 7 8 9

Rotate List [Leetcode]

```
struct ListNode* rotateRight(struct ListNode* head,
                             int k){
    struct ListNode *ptr, *ptr1;
    int count=0, num;
    if(head==NULL || head->next==NULL){
        return head;
    } ptr=head;
    while(ptr->next!=NULL){
        count++;
        ptr=ptr->next;
    } num=k%(count+1);
    while(num--){
        ptr=head;
        while(ptr->next!=NULL){
            ptr1=ptr;
            ptr=ptr->next;
            }
        ptr->next=head;
        ptr1->next=NULL
        head=ptr;
    }
    return head;
}
```

Tracing Inorder



~~Traversal~~ ~~all~~ ⓪ ~~then~~ ~~is printed~~

In Inorder left traversal happens till node->left is null then node->val is printed

~~then~~ ① ~~is printed~~

~~then~~ ② ~~is printed~~

③ 1st Node printed ⓪

Then node->right is checked, => NULL

Thus it return to previous level

and ① is printed, right is checked then ②, ③, ④ are printed

Returns to previous level ⑤ is printed & right is checked then ⑥, ⑦ is printed

Returns to previous level ⑧ is printed, right is checked then ⑨ is printed.

→ Split List into Parts

```
struct ListNode** splitListToParts(struct ListNode* head,
                                   int k, int* returnSize) {
    struct ListNode* ptr = head;
    *returnSize = k;
    int count = 0;
    while (ptr != NULL) {
        count++;
        ptr = ptr->next;
    }
    int nums = count/k, a = count%k;
    struct ListNode** L = (struct ListNode**) calloc
                          (k, sizeof(struct ListNode*));
    ptr = head;
    for(int i=0; i<k; i++) {
        L[i] = ptr;
        int segmentSize = nums + (a-- > 0 ? 1 : 0);
        for (int j=1; j<segmentSize; j++) {
            ptr = ptr->next;
        }
        if (ptr != NULL) {
            struct ListNode* next = ptr->next;
            ptr->next = NULL;
            ptr = next;
        }
    }
    return L;
}
```

→ BFS_DFS

```
#include<stdio.h>
#define MAX_VERTICES 50
typedef struct Graph_t {
    int V;
    int adj[max_VERTICES][MAX_VERTICES];
} Graph;

int DFS_V[50];
Graph* Graph_create(int V) {
    Graph* g = malloc(sizeof(Graph));
    g->V = V;
    for(int i = 0; i <= V; i++){
        for(int j = 0; j <= V; j++){
            g->adj[i][j] = 0;
        }
    }
    return g;
}

void Graph_addEdge(Graph *g, int v, int w) {
    g->adj[v][w] = 1;
    g->adj[w][v] = 1;
}

void BFS(Graph * g, int root){
    for
```

```
int visited[g->v+1];
for(int i=0; i<=g->v; i++)
    visited[i] = 0;

int queue[g->v+1];
int front=0, rear=0;
visited[root] = 1;

queue[rear++] = root;
while(front != rear){
    root = queue[front++];
    printf("%d", root);
    for(int i=0; i<=g->v+1; i++){
        if(g->adj[root][i]==1 && visited[i]){
            visited[i]=1;
            queue[rear++]=i;
        }
    }
}

int DFS(Graph *g, int root){
    for(int i=0; i<=g->v; i++){
        if(adj[root][i]==1 && DFS_V[i]==0){
            DFS_V[i]=1;
            DFS(g, i);
        }
    }
```

```
int count = 0;
for(int i = 0; i <= g->v; i++){
    if(DFS_V[i] == 1){
        count++;
    }
    return count;
}

int main(){
    Graph *g = Graph_Create(5);
    Graph_addEdge(g, 0, 1);
    Graph_addEdge(g, 0, 4);
    Graph_addEdge(g, 1, 3);
    Graph_addEdge(g, 1, 2);
    Graph_addEdge(g, 2, 3);
    Graph_addEdge(g, 4, 3);
    printf("BFS traversal: ");
    BFS(g, 0);
    int count = DFS(g, 0);
    if(count == g->V+1){
        printf("\n Graph is connected");
    }
    else { printf("\n Graph is disconnected");
    }
}
```

O/P

BFS traversal : 0 1 4 2 3
Graph is connected.

Lab 10

```
#include <stdio.h>
#define size 10
int HT[SIZE];
void hash(int a[]){
    int key
    for(int i=0; i<10; i++){
        HT[i] = -1;
    }
for(int i =0; i<10; i++){
    key = a[i]%size
    if(HT[key] == -1){
        HT[key] = a[i]; }
else { while (HT[key] != -1) {
        key = (key+1)% size;
    }
    HT[key] = a[i] }}}
void search (int num){
    int key = num % size;
    int loc = -1;
```

```
if(HT[key] == num){
    loc = key;
  }
else { while(HT[key] != num){
        key = (key+1) % size;
      }
  if(HT[key] == num) {
      loc = key;
    }}
  if(loc == -1) {
    printf(" %d is absent", num);
  } else { printf("%d is at %d", num, loc);
  }}
void main() {
  int n, num;
  printf("Enter no of employee: ");
  scanf("%d", &n);
  if(n != size){
    printf("Invalid"); }
  printf("Enter Employee ID's :\n");
  int data[10];
  for(int i=0; i<10; i++) { scanf("%d", &data[i]);
  hash(data);                                      }
  printf("Enter ID :");
  scanf("%d", &num);
  search(num); }
```

O/P

Enter no of employes : 10
Enter Employee ID's :
1001
2045
5013
4321
5033
7896
9745
8561
3245
2216
Enter ID to search : 8561
8561 is present at 08 index.

9/3/24