INDEX

NAME: Preeti . T STD _____ SEC: _____ ROLL NO: _____

S.No.	Date	Title	Page No.	Teacher's Sign/ Remarks
1		Lab practice program		
2.		Lab program - 1		
3		Lab program - 2		
4		Lab program - 3		
5		Lab program - 4 leetcode		
6.		Lab program - 5		
7		Lab program - 6		
8		Lab program - 7 Leetcode		
9		Lab program - 8 Leetcode		
10		Lab program 9 Leetcode		
11		Lab program 10 Hackerrank		

Befna Gold

1) Swapping using pointers;

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

output:

```
5 6
Before swapping a = 5 and b = 6
After swapping a = 6 and b = 5
```

Dynamic memory allocation:

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

void main()
{
    int *p1, *p2;
    int n, i;

    printf("Enter the number of array elements\t");
    scanf("%d", &n);

    p1 = (int*) malloc (n* sizeof(int));
    p2 = (int*) calloc (n, sizeof(int));

    if ( p1 == NULL && p2 == NULL)
    {
        printf("memory is not allocated");
        exit(0);
    }
    else
    {
        printf("memory is succesfully allocated");

        for (i=0; i<n; i++)
            p1[i] = i+1;
        printf("array elements in malloc are:\n");
        for (i=0; i<n; i++)
            printf("%d", p1[i]);

        for (i=0; i<n; i++)
            p2[i] = i+1;
```

```
printf ("array elements in calloc are: \n");
for (i=0; i<n; i++)
    printf ("%d", p2[i]);
}
printf ("Enter the new size of n");
scanf ("%d", &n);
p2 = (int*) realloc (p2, n*(sizeof(int)));
if (p2==NULL)
{
    printf ("memory is not allocated");
    exit(0);
}
else
{
    printf ("memory is succesfully allocated");

    for (i=0; i<n; i++)
        p2[i] = i+1;
    printf ("array elements in realloc are: \n");
    for (i=0; i<n; i++)
        printf ("%d", p2[i]);
}
free (p1);
free (p2);
}
```

output

Enter the number of array elements 5
memory is succesfully allocated
array elements in malloc are:
1 2 3 4 5
array elements in calloc are:
1 2 3 4 5

```
Enter the new size of array
8
memory is succesfully allocated
array elements in realloc are:
1 2 3 4 5 6 7 8
```

3) Stack Implementation.

```
#include <stdio.h>

int stack[100], i, j, ch = 0, n, top = -1;

void push();
void pop();
void display();
void main()
{
    printf("Enter the no. of elements in the stack");
    scanf("%d", &n);

    while (ch != 4)
    {
        printf("Choose one from the below options\n");
        printf("\n1. Push\n2. Pop\n3. Display\n4. Exit");
        printf("\n Enter your choice\n");
        scanf("%d", &ch);
        switch (ch)
        {
            case 1: push();
                    break;
            case 2: pop();
                    break;
```

```
        case 3: display();
                break;

        case 4: printf("Exit");
                break;

        default: printf("Please enter valid choice");
    }
}

void push()
{
    int val;
    if (top == n)
        printf("\n Overflow");
    else
    {
        printf("Enter the value:");
        scanf("%d", &val);
        top = top+1;
        stack[top] = val;
    }
}

void pop()
{
    if (top == -1)
        printf("Underflow");
    else
        top = top-1
}
```

```
void display()
{
    if (top == -1)
        printf ("stack is empty");
    for (i = top ; i >= 0 ; i--)
        printf ("%d \n", stack[i]);
}
```

Output:

Enter number of elements : 5

choose 1 - Push
2 - Pop
3 - Display
4 - Exit

Enter your choice
1

Enter choose 1 - Push
2 - Pop
3 - Display
4 - Exit

Enter your choice
2

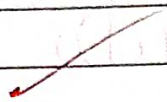
choose 1 - Push
2 - Pop
3 - Display
4 - Exit

Sept 24/12/23

Enter your choice
3

choose 1 - Push
2 - Pop
3 - Display
4 - Exit

Enter your choice
4.
exit



Page
Date / /

28/12/23 WEEK-2 LAB-1 Infix to postfix

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define size 100

void push (char st[100], char val);
int pop (char st[100]);
int pre (char op);
int top = -1;
char st[size];

void push (char st[100], char val)
{
    if (top == size-1)
    {
        printf ("Stack is full");
        exit(0);
    }
    else
    {
        top = top+1;
        st[top] = val;
    }
}
int pop (char st[100])
{
    char val;
    if (top == -1)
    {
        printf ("Stack is empty");
    }
```

Date / / Page

```
else
{
    val = st[top];
```

WEEK - 2 LAB - 2 28/12/23

Infix to postfix conversion

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define max 100
char st[100];
int top = -1;
char ele;

void push (char ele)
{
    if (top == (max) - 1)
    {
        printf("stack is full");
    }
    else
    {
        top++;
        st[top] = ele;
    }
}

char pop()
{
    if (top == 1)
    {
        printf("stack is empty");
```

```
    }
    else
    {
        top = top - 1;
        return (st[top + 1]);
    }
}
char prc (char a)
{
    if (a == '^')
    {
        return (5);
    }
    else if (a == '*')
    {
        return (4);
    }
    else if (a == '/')
    {
        return (3);
    }
    else if (a == '+')
    {
        return (2);
    }
    else if (a == '-')
    {
        return (1);
    }
    else
    return (0);
}
```

Date
Page

```
int main()
{
    char postfix[100];
    char infix[100];
    int i = 0;

    printf("Enter the infix expression \n");
    scanf("%s", infix);

    while(infix[i] != '\0')
    {
        if(isalpha(infix[i]))
        postfix[i] = infix[i];
        else if(infix[i] == '^' || infix[i] == '*' || infix[i] == '/'
                || infix[i] == '-')
        {
            if(pre(st[top]) > pre(infix[i]))
            {
                postfix[i] = pop();
                push(infix[i]);
            }
            else
            push(infix[i]);
        }
        i++;
    }
    postfix[i] = st[top];
    printf("Postfix expression is: \n");
    for(i=0; i<=strlen(infix)+1; i++)
    {
        printf("%c", postfix[i]);
    }
}
```

Output :

Enter the infix expression
A+B-C
Postfix expression is :
AB+C-

Postfix Evaluation LAB-3

```
#include <stdio.h>
int st[20];
int top = -1;

void push(int x)
{
    st[++top] = x;
}
int pop()
{
    return st[top--];
}
int main()
{
    char exp[20];
    char *e;
    int n1, n2, n3, num;
    printf("Enter the expression :: ");
    scanf("%s", exp);
    e = exp;
    while(*e != '\0')
    {
        if(isdigit(*e))
```

```
{ num = *e - 48;
  push(num);
}
else
{
    n1 = pop();
    n2 = pop();
    switch (*e)
    {
      case '+':
        n3 = n1 + n2;
        break;
      case '-':
        n3 = n2 - n1;
        break;
      case '*':
        n3 = n1 * n2;
        break;
      case '/':
        n3 = n2 / n1;
        break;
    }
    push(n3);
}
e++;
}
printf("\n The result of expression %s = %d \n\n", exp,
        pop());
```

output: Enter the expression.

1+2

The result of expression 1+2 is 3

4/01/24 LAB-3 WEEK-3

Linear Queue:

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

display()
{
    int i;
    if (front == -1)
        printf("Queue is empty\n");
    else
    {
        printf("Queue is :\n");
        for(i = front; i <= rear; i++)
            printf("%d", queue[i]);
        printf("\n");
    }
}
insert()
{
    int add;
    if (rear == MAX-1)
    printf("Queue overflow\n");
    else
    {
        if (front == -1)
        front = 0;
        printf("Insert the element");
        scanf("%d", &add);
        rear = rear + 1;
```

```
        queue[rear] = add;
    }
}
delete()
{
    if (front == -1 || front > rear)
    {
        printf("Queue underflow \n");
        return;
    }
    else
    {
        printf("Deleted Element is: %d \n", queue[fro
        front = front + 1;
    }
}
main()
{
    int choice;
    while (1)
    {
        printf("1. Insert \n 2. Delete \n 4. Exit \n");
        switch (ch)
        {
            case 1: Insert();
                    break;
            case 2: delete()
                    break;
            Case 3: display();
                    break;
            case 4: exit(1)
        }
    }
}
```

output:

1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice
1
Enter the number to be inserted into the queue.
2
Queue underflow
Enter choice : 1.
Insert an element : 6.
Enter choice : 4
Exited.

Sp. 1
28/11/23

11/01/24 WEEK-3 LAB-2

2) Circular queue

```
#include <stdio.h>
int q[50], front = -1, rear = -1, size;

void display();
void enqueue();
void dequeue();

void main()
{
    int ch;
    printf("Enter no. of elements : ");
    scanf("%d", &size);

    while (ch != 4)
    {
        printf("1. insert 2. Delete 3. Display
                4. Exit \n");
        printf("Enter your choice");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1: enqueue();
                    break;
            case 2: dequeue();
                    break;
            case 3: display();
                    break;
        }
    }
    printf("Exited");
}
```

```
void enqueue()
{
    int item;
    if ((front == rear + 1) || (front == 0 && rear == size-1))
        printf("Queue is full\n");
    else
    {
        if (front == -1)
            front = 0;
        printf("Enter element:");
        scanf("%d", &item);
        rear = (rear + 1) % size;
        q[rear] = item;
    }
}
void dequeue()
{
    int ele;
    if (front == -1)
    printf("Queue is empty\n");
    else
    {
        ele = q[front];
        if (front == rear)
        {
            front = -1;
            rear = -1;
        }
        else
        front = (front + 1) % size;
        printf("Deleted");
    }
}
```

```
void display()
{
    int i;
    if (front == -1)
      printf("Queue is empty");
    else
    {
        printf("Front = %d \t", front);
        printf("Rear = %d \n", rear);
        printf("Queue");
        for(i=front;i<=%d;i++",
        for(i = front ; i != rear ; i = (i+1)%size)
          printf("%d", q[i]);
          printf("%d \n", q[i]);
      }
}
```

output:

Enter no. of elements : 4
1. Insert 2. Delete 3. Display 4. Exit.
Enter your choice : 1
Enter element : 1.
1. Insert 2. Delete 3. Display 4. Exit.
Enter your choice : 3
1
Enter 1. Insert 2. Delete 3. Display 4. Exit
Enter your choice : 4

Linked list : WEEK - 3 LAB - 3

```
#include <stdio.h>
#include <stdlib.h>
void push();
void append();
void display();
void insert_at_pos();
struct node
{
    int idata;
    struct node *next;
};
struct node *head = NULL;

void main()
{
    int ch;
    while(ch!=6)
    {
        printf("Enter the choice \n 1. insert from \n
        2. Insert at end \n 3. Insert at particular
        pos \n 4. Display \n 5. Exit \n");
        &
        while (ch!=5)
        {
            printf("Enter choice");
            scanf("%d", &ch);
            switch(ch)
            {
                case 1: push();
                        break;
```

Date / / Page

```
            case 2: append();
                    break;
            case 3: insert_at_pos();
                    break;
            case 4: display();
                    break;
            case 5: exit(0);
          }
      }
  }
void push()
  {
      int data;
      struct node * new_node;
      new_node = struct node* malloc (sizeof(struct node));
      printf("Enter the data to be inserted \n");
      scanf("%d", &data);
      new_node -> data = data;
      new_node -> ~~data~~ next = head;
      head = new_node;
  }
void append()
  {
      int data;
      struct node * last = head;
      struct node * new_node;
      new_node = struct node * malloc (sizeof(struct node));
      printf("Enter the data \n");
      if (head == NULL)
      {
          new_node = head;
      }
```

```
else
{
    while (last -> next != NULL)
    last = last -> next;
    last = new_node;
    }
}

void insert_at_pos()
{
    int data;
    int pos;
    struct node *temp = head;
    struct node * new_node;
    new_node = struct node* malloc (sizeof (struct
                                        node));
    printf ("Enter the data to be inserted \n");
    scanf ("%d", &data);
    new_node -> data = data;
    printf (" enter the position \n");
    scanf ("%d", &pos);
    if (pos == 1)
    { new_node -> next = temp;
      new_node = head;
    }
    else
    {
        for (int i = 2; i < pos - 1; i++)
        {
            temp = temp -> next;
            new_node -> next = temp -> next;
            temp -> next = new_node;
            new_node -> next = NULL;
```

```
        }
    }
}.
void display()
{
    struct node *p = head;
    printf("The list elements \n");
    while (p->next != NULL)
    {
        printf("%d", p->data);
        p = p->next;
    }
}.
```

Output :

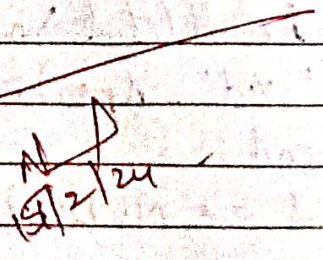
Enter the choice
1. Insert from beginning
2. Insert at end
3. Insert at particular pos
4. Display
5. Exit
Enter the choice.
1
Enter the element to be inserted
2.
Enter the choice.
1.
Enter the element to be inserted
4.
Enter the choice
4.
2
4

11/11/24

WEEK - 4 (18-01-24)

Three ways of deletions in linked list.

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

struct node
{
    int data;
    struct node *next;
};
struct node *head = NULL;

void dbegin()
{
    struct node *ptr;
    if (head == NULL)
    {
        printf("List is empty\n");
    }
    else
    {
        ptr = head;
        head = head -> next;
        free(ptr);
        printf("First element is deleted\n");
    }
}

void dend()
{
    struct node *ptr;
    struct node *ptr1;
```



```
if (head == NULL)
{
    printf ("List is empty \n");
}
else if (head -> next == NULL)
{
    free(head);
}
else
{
    ptr = head;
    while (ptr -> next != NULL)
    {
        ptr1 = ptr;
        ptr = ptr -> nat;
    }
    free (ptr);
    ptr1 -> next = NULL;
    printf (" Element at the end is deleted \n");
}
}
void dpos()
{
    Struct node *ptr;
    Struct node *ptr1;
    int pos, i;
    printf (" Enter the position from which data to
            be deleted \n");
    scanf (" %d", &pos);
    ptr = head;
    if (head == NULL)
    {
```

```
    } printf("List is empty \n");
    else if (head -> next == NULL)
    {
        free(head);
    }
    for (i=0; i<pos; i++)
    {
        ptr1 = ptr;
        ptr = ptr -> next;
    }
    ptr1 -> next = ptr -> next;
    free(ptr);
    printf("Element at the position %d is deleted \n", pos);
}

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

Date / / Page

```
void main()
{
    int n, i, data;
    printf("Enter the number of elements in linked
            list \n");
    scanf("%d", &n);
    printf("Enter the data to be inserted \n");
    for (i=0; i<n; i++)
    {
        struct node * last = head;
        struct node * new_node;
        new_node = (struct node*) malloc (sizeof(
                struct node));
        scanf("%d", &data);
        new_node -> data = data;
        new_node -> next = NULL;
        if (head == NULL)
        {
            head = new_node;
        }
        else
        {
            while (last -> next != NULL)
            {
                last = last -> next;
            }
            last -> next = new_node;
        }
    }
    int ch;
    printf("Enter \n 1: Delete from beginning
\n 2: Delete at particular position \n 4:
Display elements \n 5: Exit \n");
```

```
while (ch != 5)
{
    printf ("Enter your choice \n");
    scanf ("%d", &ch);
    switch (ch)
    {
        case 1 : dbegin();
                 break;
        case 2 : dend();
                 break();
        case 3 : dpos();
                 break;
        case 4 : display();
                 break;
    }
}
}
```

Output:

```
Enter the number of elements in linked list
5
Enter the data to be inserted
1  2  3  4  5
Enter
1: Delete from begin
2: Delete at end
3: Delet at pos
4: Display
5: Exit
Enter your choice
1
```

First element is deleted
Enter your choice
2
Element at the end is deleted
Enter your choice
3
Enter the position from which data to be deleted
1
Element at the position 1 is deleted
Enter your choice
4
2 → 4 →
Enter your choice
5

Leetcode reversing linked list (25/01/24)

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

struct ListNode (struct Listnode * head, int left, int right)

struct ListNode * ptr = head;
struct ListNode * ptr1 ;
struct ListNode * front ;
for (int i = 1 ; i < left ; i++)
{
    ptr = head;
    head = head -> next;
    ptr 1 = head -> next;
}
struct node * back = head ;

for ( int i = left ; i < right ; i++)
{
    front = ptr 1 -> next ;
    ptr 1 -> next = back ;
    back = ptr 1 ;
    ptr 1 = front ;
}

ptr -> next = back;
head -> next = front ;

return head;
```

15/2/24

LAB 6. (25/01/24)

2) Concatenation, sorting, reversing linked list

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

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

struct node * head = NULL;
void main()
{
    int n, i, data;
    printf("Enter the no. of elements in first
            linked list\n");
    scanf("%d", &n);
    for(i=0; i<n; i++)
    {
        struct node * last1 = head1;
        struct node * new_node1;
        new_node1 = (struct node*) malloc(sizeof
                    (struct node));
        scanf("%d", &data);
        new_node1 -> data = data;
        new_node -> next = NULL;
        if (head1 == NULL)
        {
            head1 = new_node1;
        }
```

```
else
{
    while (last1 -> next != NULL)
    {
        last1 = last1 -> next;
    }
    last1 -> next = new_node;
}

printf (" Elements after sorting \n");
void sort();

printf (" Elements after reversing \n");
void reverse();
struct node * head1 = NULL; struct node * last2 = head1;
printf (" Enter the no. of elements in second
        linked list to concatenate \n");
int n;
scanf ("%d", &n);
printf (" Enter the data elements \n");
for (i = 0; i < n; i++)
{
    struct node * last2 = head;
    struct node * new_node2;
    new_node2 = (struct node *) malloc (sizeof
                (struct node));
    scanf ("%d", &data);
    new_node2 -> data = data;
    new_node2 -> next = NULL;
    if (head2 == NULL)
    {
        head2 = new_node2;
    }
```

```
    else
    {
        while (last2 -> next != NULL)
        {
            last = last2 -> next;
        }
        last2 -> next = new_node2;
    }
}
printf(" Elements after concatenation are \n");
void concat();
}

void sort()
{
    struct node *curr = head;
    struct node *ptr = NULL;
    int temp;
    while (curr != NULL)
    {   ptr = curr -> next;
        if (
            while (ptr != NULL)
            {
                if (curr -> data > ptr -> data)
                {
                    temp = curr -> data;
                    curr -> data = ptr -> data;
                    ptr -> data = temp;
                }
                ptr = ptr -> next;
            }
            curr = curr -> next;
        }
```

```
struct node * node = head1;
if (head == NULL)
{
} printf (" List is empty \n");
else
{
    while (node != NULL)
    {
        printf ("%d -> ", node -> data);
        node = node -> next;
    }
    printf ("\n");
}

void reverse ()
{
    struct node * ptr prev = NULL;
    struct node * ptr = NULL;
    while (head1 != NULL)
    {
        ptr = head1 -> next;
        head1 -> next = prev;
        prev = head1;
        head1 = ptr;
    }
    head1 = prev;
}

struct node * node = head1;
if (head == NULL)
{
```

```
    printf(" List is empty \n");
  }
  else
  {
    while(node != NULL)
    {
      printf("%d -> ", node -> data);
      node = node -> next;
    }
    printf("\n");
  }
}

void concat()
{
  struct node * temp ;

  if(head2 == NULL)
  {
    struct node * node = head2;
    if(head2 == NULL)
    {
      printf(" List is empty \n");
    }
    else
    {
      while(node != NULL)
      {
        printf("%d -> ", node -> data);
        node = node -> next;
      }
```

LAB 6. (25/01/24)

Stack Implementation

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

struct node
{
    int data;
    struct node *next;
};
struct node *head = NULL;

void main()
{
    int ch;
    printf(" Enter the choice.\n");
    scanf("%d", &ch);
    printf(" Enter 1: pop \n 2: push");
    while (ch != 4)
    { switch (ch)
      {
          case 1: pop();
                  break;
          case 2: push();
                  break;
          case 3: display();
                  break;
          case 4: exit();
      }
    }
}
```

```
void push()
{
    int n, i, data;
    printf("Enter the data\n");
    struct node *last = head;
    struct node *new_node;
    new_node = (struct node*)malloc(sizeof
                (struct node));
    scanf("%d", &data);
    new_node->data = data;
    new_node->next = NULL;
    if(head == NULL)
    {
        head = new_node;
    }
    else
    {
        while(last->next != NULL)
        {
            last = last->next;
        }
        last->next = new_node;
    }
}
```

13/2/24

```
void pop()
{
    struct node *ptr;
    struct node *ptr1;
    if(head == NULL)
    {
        printf("List is empty\n");
```

```
else if (head -> next == NULL)
{
    free(head);
}
else
{
    ptr = head;
    while (ptr -> next != NULL)
    {
        ptr1 = ptr;
        ptr = ptr -> next;
    }
    free(ptr);
    ptr1 -> next = NULL;
    printf(" Element at * is poped \n");
}
}

void display()
{
    struct node * node = head;
    if (head == NULL)
    {
        printf("List is empty \n");
    }
    else
    {
        while (node != NULL)
        {
            printf ("%d ->", node -> data);
            node = node -> next;
        }
        printf("X\n");
    }
}
```

QUEUE Implementation

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

struct node *r
{
    int data;
    struct node * next;
};
struct node * head = NULL;

void main()
{
    printf (" Enter 1: enque \n 2: dequeue \n 3: display
    int ch;
    printf (" Enter the choice \n");
    scanf ("%d", &ch);
while(ch!=4)
{   switch (ch):
    {
        case 1: enque();
                break;

        case 2: deque();
                break;

        case 3: display();
                break;
        case 4: exit();
    }
}
}
```

```
void enqueue()
{
    int data;
    printf("Enter the data to be inserted \n");
    struct node * last = head;
    struct node * new_node;
    new_node = (struct node *) malloc(sizeof(
                struct node));
    scanf("%d", &data);
    new_node -> data = data;
    new_node -> next = NULL;
    if (head == NULL)
    {
        head = new_node;
    }
    else
    {
        while (last -> next != NULL)
        {
            last = last -> next;
        }
        last -> next = new_node;
    }
}
void dequeue()
{
    struct node * ptr;
    if (head == NULL)
    {
        printf("List is empty\n");
    }
    else
```

```
    {
        ptr = head;
        head = head -> next;
        free(ptr);
        printf(" First element is deleted\n");
    }
}
void display()
{
    struct node * node = head;
    if (head == NULL)
    {
        printf(" List is empty\n");
    }
    else
    {
        while (node != NULL)
        {
            printf("%d", node -> data);
            node = node -> next;
        }
        printf("\n");
    }
}
```

Output:

```
Enter
1 : enqueue
2 : dequeue
3 : display
4 : exit
Enter the choice
1
Enter the data to be inserted
```

1
Enter the choice
1
Enter the data to be inserted
2
Enter the choice
1
Enter the data to be inserted
3
Enter the choice
3
1 → 2 → 3 →
Enter the choice
Enter 2
First element is deleted
Enter the choice
3
2 → 3 →
Enter the choice
4

output for stack implementation:

Enter 1: pop
2: push
3: display
4: exit
Enter the choice
2
Enter the data to be inserted
1
Enter the choice
2

Enter the data to be inserted
2
Enter the choice
2
Enter the data to be inserted
3
Enter the choice
1
Elementent at the end is deleted
Enter the choice
4.

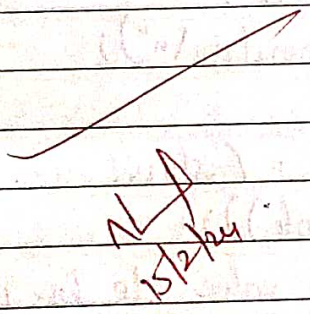
LeetCode:

Minstack:

```
typedef struct {
  int size;
  int top;
  int * s;
  int * minstack;
} minstack

minstack * minstackCreate()
{
   minstack *st = (Minstack *) malloc (sizeof minstack

   if (st == NULL)
   {
      printf (" memory allocatn failed ");
      exit(0);
   }
   st -> size = 5;
   st -> top = -1;
   st -> s = (int *) malloc (st -> size * sizeof (int));
```

```
st -> minstack = (int *) malloc (st -> size * sizeof(int))
if (st -> s == NULL)
{
    printf ("memory allocation failed");
    free (st -> s);
    free (st -> minstack);
    exit(0);
}
return st;
}

void minstackpop (Minstack * obj)
{
    int val;
    if (obj -> top == -1)
    {
        printf ("underflow");
    }
    else
    val = obj -> s[obj -> top];
    obj -> top--;
    printf ("%d is popped \n", value);
}

int minstack top (Minstack * obj)
{
    int value = -1;
    if (obj -> top == -1)
    {
        printf ("underflow \n");
        exit(0);
    }
```

```
else
    return obj->minstack[obj->top];

void minstackFree(MinStack* obj)
{
    free(obj->s);
    free(obj->minstack);
    free(obj);
}
```



15/2/24

LAB 7 , 3/02/24

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

struct node
{
    int val;
    struct node * prev;
    struct node * next;
};
struct node *head = NULL;
void insert_left()
{
    int pos;
    struct node * ptr1 = head;
    printf ("Enter the position\n");
    scanf ("%d", &pos);
    struct node * new_node = (struct node *) malloc
        (sizeof (struct node));
    printf ("Enter the value to be inserted\n");
    scanf ("%d", &new_node -> val);
    if (head == NULL)
    {
        new -> prev = NULL;
        new -> next = NULL;
        head = new;
        printf ("Node inserted\n");
    }
    else
    {
        for (i = 0; i < pos - 1; i++)
        {
            ptr = ptr -> next;
            new -> prev = ptr -> prev;
```

15/02/24

1) Create a binary search tree, perform pre-order, post-order and in order and display the contents.

```
#include

ptr -> prev -> next = new;
ptr -> prev = new;
printf("Node inserted\n");
}

void delete()
{
    int val;
    printf("Enter the value:");
    scanf("%d", &val);
    struct node * ptr = head;
    if(head -> data == val)
    {
        head = ptr -> next;
        free(ptr);
        printf("Node deleted\n");
        return;
    }
    while(ptr -> data != val)
    {
        ptr = ptr -> next;
        if(ptr -> next == NULL)
        {
            ptr -> prev -> next = NULL;
            free(ptr);
            printf("Node deleted\n");
            return;
        }
    }
```

```
    ptr -> prev -> next = ptr -> next
    ptr -> next -> prev = ptr -> prev;
    free(ptr);
    printf("Node deleted \n");
}

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

void main()
{
    int ch;
    printf("Enter 1. Insert \n 2. Delete \n 3. Displ
            4: Exit \n");

    while(ch != 4)
    {
        printf("Enter choice:");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1: insert_left();
                    break;
```

```
        case 2 : delete() ;
                 break;

        case 3 : display();
                 break;
    }
}
```

output :

```
Enter:
1 : insert
2 : delete
3 : display
4 : exit
Enter your choice
1
Enter the pos
1
Enter the val
1
```

15/2/24

Enter your choice
1
Enter the pos
1
Enter the data val.
2
Enter your choice
1
Ent the pos
1
Enter the val
3
Enter your choice.
1
Enter the pos.
2
Enter the val
4
Enter the data to delete.
Ent. deleted

LAB - 8 15/02/24

1) Create a binary search tree perform pre-order post-order and in-order and display contents.

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

struct node
{
    int data;
    struct node * left;
    struct node * right;
};
struct node * root = NULL;

void create()
{
    struct node* new node = (struct node*) malloc (sizeof(struct node));
    printf("Enter the data to be inserted \n");
    scanf("%d", &new node -> data);
    new node -> left = NULL; new_node -> right = NULL;
    if (root == NULL)
    {
        new_node -> left = NULL;
        new_node -> right = NULL;
        root = new_node;
    }
    else
    {
        ptr = root;
        while (ptr -> left != NULL || ptr -> right != NULL)
        {
            if (new node -> data > ptr -> data)
            {
```

```
        if (ptr -> right != NULL)
        {
            ptr = ptr -> right;
        }
        else
        {
            break;
        }
        else
        {
            if (ptr -> left != NULL)
            {
                ptr = ptr -> left;
            }
            else
            {
                break;
            }
        }
    }
    if (new_node -> data > ptr -> data)
    {
        ptr -> right = new_node;
    }
    else
    {
        ptr -> left = new_node;
    }
}
```

```
void pre_order (struct node * ptr)
{
    struct node * trav = ptr;
    if (ptr != NULL)
    {
        printf ("%d", ptr -> data);
        pre_order (ptr -> left);
        pre_order (ptr -> right);
    }
}

void inorder (struct node * ptr)
{
    struct node * trav = ptr;
    if (ptr != NULL)
    {
        inorder (ptr -> left);
        printf ("%d", ptr -> data);
        inorder (ptr -> right);
    }
}

void post_order (struct node * ptr)
{
    struct node * trav = ptr;
    if (ptr != NULL)
    {
        post_order (ptr -> left);
        post_order (ptr -> right);
        printf ("%d", ptr -> data);
    }
}
```

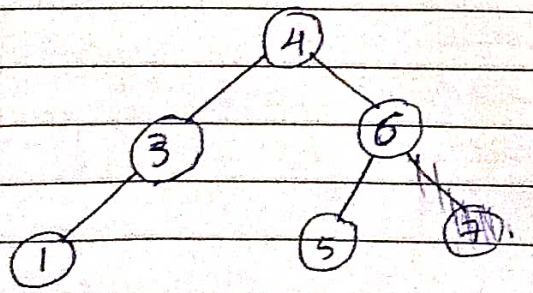
```
void main()
{
    printf(" Enter \n 1. Create \n 2. Pre-order \n 3. In-order
        4. post-order \n 5. Exit \n");
{   int ch;
    do {
        printf (" Enter your choice \n");
        scanf (" %d ", &ch);
        Switch (ch)
        {
            case 1 : create();
            break;
            case 2 : pre-order (root);
            break;
            case 3 : inorder (root);
            break;
            case 4 : post-order (root);
            break;
        }
    } while (ch != 5);
}
```

output:

Enter
1. Create
2. Pre-order
3. In-order
4. post-order
5. EXIT

Enter your choice
1

Enter the data to be inserted
4
Enter the choice
1
Enter the data to be inserted
6
Enter your choice
1
Enter the data to be inserted
1
Enter the choice
1
Enter the data to be inserted
3
Enter the choice
1
Enter the data to be inserted
5
Enter your choice
3
1 3 4 5 6 Enter your choice
2
4 1 3 6 5 Enter your choice
4
3 1 5 6 4 Enter your choice
5



15/02/24

Lect Code - 3

```
struct ListNode ** splitListToParts (struct ListNode *
head, int K, int * returnSize)
{
    struct ListNode ** result = (struct ListNode * *)
        malloc (K * sizeof (struct ListNode *));
    Struct ListNode * ptr = head;
    int i = 0, length = 0;
    int r, count = 0;

    while (ptr != NULL)
    {
        length ++;
        ptr = ptr -> next;
    }
    r = length % K;
    int partsize = length / K;

    for (int j=0; j < curr_part-1 ; j++)
    {
        if (head != NULL)
        {
            head = head -> next;
        }
    }
    if (head != NULL)
    {
        struct ListNode * temp = head;
        head = head -> next;
        temp -> next = NULL;
    }
```

```
    }

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

15/02/24

LeetCode - 4

Rotate List

```
Struct ListNode * rotateRight (struct ListNode *
  head, int K)
{
    struct ListNode * ptr = head;
    int count = 1;
    if (head == NULL || head -> next == NULL || K == 0)
        return head;
    else
    {
        while (ptr -> next != NULL)
        {
            ptr = ptr -> next;
            count ++;
        }
        if (K % count == 0)
            return head;
        ptr -> next = head;
        for (int i = K % count ; i < count ; i++)
        {
            ptr = ptr -> next;
        }
        head = ptr -> next;
        ptr -> next = NULL;
        return head;
    }
}
```

Bafna Gold
Date: Page:

29/02/24 HASH TABLE

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

#define MAX_MEMORY 10

int hash[Max] = {0};
int status[Max] = {0};

void initial{int size}
{
    for(int i = 0; i<size; i++)
    {
        status[i] = 0;
    }
}

int hashFun(int key, int size)
{
    return key % size;
}

int linProbe(int h, int a, int size)
{
    return (h + a) % size;
}

void insertEmployee(int size, int key)
{
    int h = hashFun(key, size);
    int index = hashValue;
    int index =
    int attempt = 1;
```

```
while (status[index] == -1)
{
    index = linearProbe(ha, attempt, size);
    attempt++;
}
hashTable[index] = key;
status[index] = 1;

void display(int size)
{
    for(int i = 0; i < size; i++)
    {
        if (status[i] == -1)
        {
            printf("%d\n", hashTable[i]);
        }
        else
        {
            printf("Empty");
        }
    }
}

int main()
{
    int m = MAX;
    initializeHashTable(m);

    int n;
    printf("Enter the number of employee records:");
    scanf("%d", &n);
```

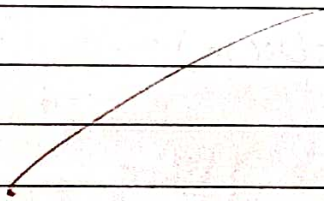
```
int Keys[n];

printf("Enter the employee Keys: \n");
for (int i=0; i<n; i++)
{
    scanf("%d", &Keys[i]);
}
for (int i=0; j<n; i++)
{
    scanf("%d", &Keys[i]);
}
for (int i=0; i<n; i++)
{
    insertEmployee(m, Keys[i]);
}
displayHashTable(m);
return 0;
}
```

Output:

```
Enter the number of employee records : 5
Enter the employee keys:
Employee 1 : 1
Employee 2 : 3
Employee 3 : 5
Employee 4 : 34
Employee 5 : 35
```

Hash Table

Index	Key
0	
1	1
2	
3	3
4	34
5	5
6	35
7	
8	
9	



22/2/24 LAB-9

Bafna Gold
Date: Page:

BFS & DFS traversals

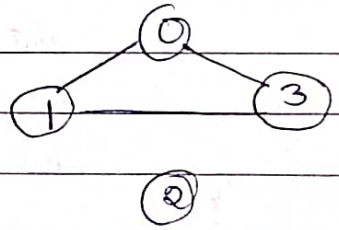
```
#include <stdio.h>
int queue[100];
int f = 0, r = 0
int a[10][10];

void enqueue (int var)
{
    queue[r] = var;
    r++;
}
void dequeue()
{
    f++;
}
void bfs (int n)
{
    int visit[10] = {0};
    enqueue(0);
    visited[0] = 1;
    while (f != r)
    {
        int curr = queue[f];
        printf("%d", curr);
        dequeue();
        for (int i = 0; i < n; i++)
        {
            if (a[curr][i] && !visited[i])
            {
                visited[i] = 1;
                enqueue[i];
```

```
        }
      }
    }
}

void dfs(int a[][10], int n, int start,
         int visited2[10])
{
    visited2[start] = 1;
    printf("%d", start);
    for(int i=0; i<n; i++)
    {
        if(a[start][i] && !visited2[i])
        {
            dfs(a, n, i, visited2);
        }
    }
}

void main()
{
    int n;
    int start = 0;
    int visited2[10];
    printf("Enter no. of vertices:");
    scanf("%d", &n);
    printf("Enter adjacency matrix :\n");
    for(int i=0; i<n; i++)
    {
        for(int j=0; j<n; j++)
        {
            scanf("%d", &a[i][j]);
        }
```

```
printf("BFS traversal:\n");
bfs(n);

for(int i=0; i<10; i++)
{
    visited2[i]=0;
}
printf("DFS traversal:\n");
dfs(a, n, start, visited2);
}.
```

Output:

Enter no of vertices : 4
Enter adjacency matrix
0 1 0 1
1 0 0 1
0 0 0 0
1 1 0 0



BFS traversal:
1 0 3
DFS traversal:
0 1 3
Disconnected graph

Hacker rank (trees)

```
typedef struct TreeNode
{
    int data;
    struct TreeNode * left;
    struct TreeNode * right;
} TreeNode;

void inOrderTraversal (TreeNode * root , int * result,
int * index)
{
    if ( root == NULL)
    {
        return;
    }
    inOrderTraversal (root -> left, result, index);
    result [(* index )++] = root -> data;
    inOrder Traversal (root -> right, result, index);
}

void swapSubtrees (TreeNode * root , int k, int depth)
{
    if (root == NULL)
      return;
    if (depth % k == 0)
    {
        TreeNode * temp = root -> left;
        root -> left = root -> right;
        root -> right = temp;
    }
    swapSubtrees ( root -> left , k, depth+1);
    swapSubtrees (root -> right, depth+1);
}
```

```
TreeNode * buildTree (int indexes_rows, int index_
columns, int * * indexes)
{
    TreeNode * root = (TreeNode *) malloc (sizeof
    (TreeNode));
    root -> data = 1;
    root -> left = NULL;
    root -> right = NULL;
    TreeNode * nodes [indexes_rows + 1];
    nodes[1] = root;

  for (int i = 0; i < indexes_rows; i++)
  {
      TreeNode * curr = node[i+1];
      if (indexes[i][0] != -1)
      {
          curr -> left = (TreeNode *) malloc (sizeof TreeNode
          curr -> left -> data = indexes[i][0];
          curr -> left -> left = NULL;
          curr -> left -> right = NULL;
          nodes[index[i][0]] = curr -> left;
      }
  }
  return root;
}
int ** swapNodes (int indexes_rows, int
index_columns, int ** indexes, int queries_count,
int * queries, int * result_rows, int * result_
columns)
{
  int ** result = (int **) malloc (queries_count
    * sizeof (int *));
```

```
*returnSize = queriesCount;
*returnColumnSizes = indexesRowSize;

TreeNode* root = buildTree(indexesRowSize, indexesColSize, indexes);

for (int i = 0; i < queriesCount; i++)
{
    int k = queries[i];
    swapSubtrees(root, k, 1);

    int* traversal = (int*)malloc(indexesRowSize * sizeof(int));
    int index = 0;
    inOrderTraversal(root, traversal, &index);
    result[i] = traversal;
}
return result;
}
```