LAB-1

Date __/__/__
Page _____

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

void main()
{
    int ch;
    while(1)
    {
        printf("Operations on the stack :\n");
        printf(" 1.Push the element \n 2.Pop the
          element \n 3.show \n 4.End \n");
        scanf("%d", &ch);

    switch(ch)
    {
    case 1:
        push();
        break;
    case 2:
        pop();
        break;
    case 3:
        show();
        break;
    case 4:
```

```
    }
  }
}

void push()
{
    int n;
    if (top == SIZE-1)
    {
      printf("Overflow \n");
    }
    else
    {
      printf("Enter the element to be added
              in the stack : \n");
      scanf("%d", &n);
      top = top + 1;
      inp_array[top] = n;
    }
}

void pop()
{
  if (top == -1)
  {
    printf("Underflow \n");
  }
  else
  {
    printf("Popped element %d \n", inp_array
                                    [top]);
    top = top - 1;
  }
}
```

```
void show()
{
  if (top==-1)
  {
    printf("Underflow");
  }
  else
  {
    int i;
    printf("Element in the stack are: \n");
    for (i=top; i>=0; --i)
        printf("%d\n", inp_array[i]);
  }
}
```

Output.

Operations on the stack:
1. Push the element
2. Pop the element
3. Show
4. End
Enter the choice
1.
Enter the element to be added in the stack:
2.
Operations on the stack:
1. Push the element
2) Pop the element
3 show
4 End
Enter the choice
1.

Enter the element to be added in the st
4
Operations on the stack:
1. Push the element
2. Pop the element
3. Show
4. End.
Enter the choice:
1.
Enter the element to be added in the st
5
Operations on stack:
1 Push the element
2. Pop the element
3 Show
4 End

LAB-2 Infix to Postfix:

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#define pf printf
#define sf scanf
#define MAX 100

int stack[MAX];
char postfix[MAX], infix[MAX]
int top = -1;

void push(char);
char pop();
int space(char);
void inToPost();
int precedence(char);
int isempty();
void print();
int post_eval();

int main()
{
    int result;
    pf("Enter the infix expression \n");
    gets(infix);

    inToPost();
    print();
    result = post_eval();
    pf("\n\n\n The value of the above post
        fix expression is: %d", result);
    return 0;
```

```
void inToPost()
{
    int i,j=0;
    char symbol, next;
    for(i=0; infix[i] != '\0'; i++)
    {
        if(!space(infix[i]))
        {
            symbol = infix[i];
            switch(symbol){
                case '(':
                    push(symbol);
                    break;
                case ')':
                    while(next != pop() != '(')
                    postfix[j++] = next;
                    break;
                case '+':
                case '-':
                case '/':
                case '*':
                case '^':
                while(!isempty() && precenden(stack[top]
                    >= precedence[symbol])
                    postfix[j++] = pop();
                push(symbol);
                break;
                default:
                    postfix[j++] = symbol;
                    break;
            }
        }
    }
```

```
while (!isempty())
{
        postfix[j++] = symbol pop();
}
postfix[j] = '\0';
}

void push(char c)
{
    stack[++top] = c;
}
char pop()
{
    return stack[top--];
}
int isempty()
{
    if (top == -1)
        return 1;
    else
        return 0;
}

int precedence(char symbol)
{
    switch(symbol){
        case '^' : return 3 ; break;
        case '+' :
        case '-' :
                    return 1 ; break ;
        case '/' :
        case '*' : return 2; break;
        default : return 0;
    }
}
```

```
void print()
{
    int i=0
    pf("\n The postfix expression will be\n");
    while (postfix[i])
    {
    pf("%c", postfix[i++]);
    }
    pf("\n");
}

int space(char c){
    if (c==' ' || c=='\t')
        return 1;
    else
        return 0;
}

int post_eval()
{
    int i, a, b;
    for(i=0; i<strlen(postfix); i++)
    {
        if(postfix[i]>='0' && postfix[i]<='9')
            push(postfix[i] - '0');
        else{
            a= push(); pop();
            b= pop();
            switch(postfix[i]):
                case '+': return push(a+b);
                          break;
                case '-': return push(b-a);
                          break;
```

```
case '/'
case '*'
case '^'
    }
}
pop();
}
```

Output

Enter
3*5

The
35*2

The

```
        case '/' : push(b/a);
                   break;
        case '*' : b*a; push(b*a)
                   break;

        case '^' : push(pow(b,a)); break;
    }
}
}
pop();
}
```

Output:

Enter the infix expression:
3*5 -2+1/2

The postfix expression will be
35*2 -12/+

The value of the above post fix expression is: 13

LAB-3 Linear Queue.

```
#include<stdio.h>
#define SIZE 30

int queue[size];
int front = -1;
int rear = -1;

void insert (int a)
{
   if (rear == SIZE -1)
   {
      printf("Queue Overflow\n");
      return;
   }
   else
   {
      if (front==-1)
         front=0;
      queue[++rear] = a;
   }
}
void delete()
{
   if (front==-1||front>rear)
   {
      printf("Queue Empty\n");
   }
   else
   {
      front++;
   }
}
```

```
void dis
{
   i
   if
   {
   }
}
voi
{
```

```
void display()
{
    int i;
    if (front == -1)
    {
        printf("Queue Empty\n");
        return;
    }
    printf("Queue : ");
    for(i=front ; i<=rear ; i++)
    {
        printf("%d", queue[i]);
    }
}

void main()
{
    int choice;
    int a;
    while (1)
    {
        printf("\n 1. Insert \n 2. Delete \n 3. Display
        \n Choice:");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1 : printf("Enter an element");
                 scanf("%d", &a);
                 insert(a);
                 display();
                 break;
```

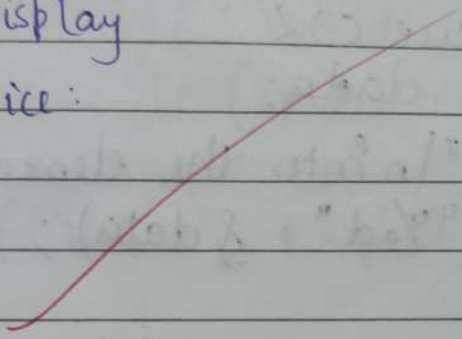
```
        case 2 : delete ();
                 display ();
                 break;

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

Output:

```
1. Insert
2. Delete
3. Display
Choice: 1
Enter an element: 24
Queue : 24
1. Insert
2. Delete
3. Display
Choice : 1
Enter an element: 25
Queue : 24 25
1. Insert
2. Delete
3. Display
Choice: 1
Enter an element : 99
Queue : 24 25 99
1. Insert
2. Delete
3. Display
Choice : 2
```

Queue: 25 99
1. Insert
2. Delete
3. Display
Choice: 3
Queue: 25 99
1. Insert
2. Delete.
3. Display
Choice:



LAB 7

```
#include <stdio.h>
#define pf printf
#define sf scanf
#define MAX 100
int cq [MAX];
int front = -1, rear = -1;

void enqueue (){
    int data;
    pf("\n Enter the element: ");
    sf("%d", &data);

    if (front == -1 && rear == -1){
        front = rear = 0;
        cq [rear] = data;
    }
    else if ((rear+1) % MAX == front)
        pf("\n Queue Overflow");
    else{
        rear = (rear+1) % Max;
        cq [rear] = data;
    }
}
void dequeue (){
    if (front == -1 && rear == -1){
        pf("\n Queue Empty")
    }
    else if (front == rear){
        front = rear = -1;
    }
    else{
        front = (front+1) % MAX;
```

```
}
}

void display()
{
    int i=front;
    pf("\n The circular queue is : \n");
    while(i!=rear){
        pf("\t %d", cq[i]);
        i=(i+1)%MAX;
    }
    pf("\t %d", cq[i]);
}
int main()
{
    int ch;
    while(1){
        pf("\n --- ENTER --- \n 1. Insert \n
            2. Deletion \n 3. Display \t");
        sf("%d", &ch);
        switch(ch){
            case 1: enqueue();
                    display();
                    break;
            case 2: dequeue();
                    display();
                    break;
            case 3: display();
                    return0;
        }
    }
}
```

Output:

--- Enter ---

1. Insertion
2. Deletion
3. Display 1.

Enter the element: 2

The circular queue is:

2.

--- Enter ---

1. Insertion
2. Deletion
3. Display 1.

Enter the element: 4

The circular queue is:

2 4.

--- Enter ---

1. Insertion
2. Deletion
3. Display 1.

Enter the element: 6

The circular queue is:

2 4 6

--- Enter ---

1. Insertion
2. Deletion
3. Display 2.

The circular queue is:

4 6

--- Enter ---

1. Insertion
2. Deletion
3. Display 3

The circular queue is:

4 6

LAB 4

```
#include <stdio.h>
#include <stdlib.h>
#define pf printf
#define sf scanf

struct node {
        int data;
        struct node * next;
};
struct node *head, *newnode, *temp, *last;

void push()
{
  newnode = (struct node*) malloc
                (sizeof (struct node));
  pf("Enter the data of the node to
    be inserted at beg : ");
  sf("%d", &newnode -> data);
  newnode -> next = head;
   head = newnode;
  pf("\n--- The element %d has been
        inserted at beg ---", newnode ->
                                data);
}
void append ()
{
  newnode = (struct node*) malloc
                (size of (struct node));
  pf("Enter the data of node : ");
  sf("%d", &newnode -> data);
  newnode -> next = NULL
  temp = head;
  while (temp -> next != NULL)
```

```
{
    temp = temp->next;
}
temp->next = newnode;
pf("\n The element %d has been inserted at
    the end :", newnode->data);
}

void insert_pos()
{
    int pos, i=0;
    newnode = (struct node*) malloc(sizeof
                (struct node));
    pf("Enter the data of the node :");
    sf("%od", &newnode->data);
    pf("\nEnter the pos :");
    sf("%od", &pos);
    temp = last->head;

    for(i=0; i<pos; i++)
    {
        last = temp;
        temp = temp->next;
    }
    last->next = newnode;
    newnode->next = temp;
}
void display()
{
    pf("The linked list you have created
    is: \n");
    temp = head;
```

```
while(temp != NULL)
{
    pf("%d\t", temp->data);
    temp = temp->next;
}
}

int main()
{
    int ch;
    while(1){
    pf("\n\n Enter your choice : \n
      1. Insert at beginning \n 2. Insert at end
    \n 3. Insert at a pos \n 4. Display: ");
    sf("%d", &ch);
    switch(ch){
    case 1 :
            push();
            display();
            break;
    case 2: append();
            display();
            break();
    case 3: insert_pos();
            display();
            break;
    case 4: display();
            return 0;
    }
    }
}
```

Output:

Enter your choice:
1. Insert at beginning
2. Insert at End
3. Insert at a pos
4. Display
Enter the data of the appending node: 6.

The element 6 has been inserted at the end: The linked list you have created is:
2 4 6

Enter your choice:
1. Insert at beginning
2 Insert at End
3. Insert at a pos
4. Display :3
Enter the data of the node: 0.

Enter the pos: 2
The linked list you have created is:
2 4 0 6

Enter your choice
1. Insert at beginning
2 Insert at End
3. Insert at a pos
4. Display : 4.
The linked list you have created is:
2 4 0 6

LAB-5

```
#include <stdio.h>
#include <stdlib.h>
#define pf printf
#define sf scanf

struct node {
    int data;
    struct node *next;
};
struct node *head, *newnode, *temp, *temp1;

void create()
{
  newnode = (struct node *) malloc (sizeof (struct node));

pf("Enter the data of the node to be inserted
 at beginning : ");
sf ("%d", &newnode->data);
newnode->next = head;
head = newnode;
pf("\n --- The element %d has been inserted
    at beg ---", newnode->data);
pf("\nThe linked list you have created is : \n");
temp = head;
while ( temp != NULL)
{
    pf("%d\t", temp->data);
    temp = temp->next
}
}
```

```
void delete_at_beg()
{
    if (head == NULL)
    {
        pf("\n The linked list is empty");
    }
    else if (head->next == NULL)
    {
        head = NULL;
        pf("\n The only node is deleted,
            the list is now empty");
    }
    else
    {
        head = temp->next;
        free(temp);
    }
}

void delete_at_end()
{
    if (head == NULL)
    {
        pf("\n The Linked list is empty");
    }
    else if (head->next == NULL)
    {
        head = NULL;
        pf("\nThe only node is deleted,
            the list is now empty");
    }
    else
    {
        while (temp->next != NULL)
        {
            temp1 = temp
            temp = temp->next;
        }
        temp1->next = NULL;
```

```
    free(temp);
}

void delete_at
{
    temp = he
    int pos
    if (hea
        pf(
    }
    else i
    {
    else
    {
    fo
```

```
    free(temp);
}

void delete_at_pos()
{
    temp = head;
    int pos, i;
    if (head == NULL) {
        pf("\n The linked list is
            empty");
    }
    else if ( head->next == NULL)
    {
        head = NULL;
        pf("\n The only node is deleted,
            The list is now empty");
    }
    else {
        pf("\n Enter the position of element
            to be deleted:");
        sf("%d", &pos);

        for( i=0; i<pos; i++)
        {
            temp1 = temp;
            temp = temp->next;
            if (temp == NULL) {
                pf("\n There are less than %d
                    elements in the list", pos);
                return 0;
            }
        }
        temp1->next = temp->next
        free(temp);
```

```
}

void display()
{
    pf("The linked list you have created
        is : \n");
    temp = head;
    while (temp != NULL)
    {
        pf("%d\t", temp -> data);
        temp = temp -> next;
    }
}

int main()
{
while(1){   int ch;
    pf("\n --- Menu --- 1. Create \n 2. Delete
      at beg \n 3. Delete at end \n
       4. Delete at pos \n 5. Display :");
 sf("%d", &ch);
  switch (ch){
     case 1 : create();
              break;
     case 2 : delete_at_beg();
              break;
     Case 3 : delete_at_end();
              break;
     case 4 : delete_at_pos();
              break;
     Case 5 : display();
              break; return 0;
          }
    }
}
```

Output :

--- Menu
1. Create
2. Delete-
3. Delete-a
4. Delete-
5. Displa

Enter th
at the

The ele
the
The li

2

---
1. Cre
2. De
3. De
4.
5. D

Ente
at

The

Th

Output:

--- Menu ---
1. Create
2. Delete-at-beg
3. Delete-at-engl
4. Delete-at-pos
5. Display 1

Enter the data of the node to be inserted at the beginning : 20

The element 20 has been inserted at the beginning of node:
The linked list you have created is:
20

--- Menu ---
1. Create
2. Delete-at-beg
3. Delete_at-end
4. Delete-at pos
5. Display 1.

Enter the data of the node to be inserted at the beginning : 25.

The element 25 has been inserted at the beginning of node.
The linked list you have created is.
25 20

---Menu---

1. Create
2. Delete-at-beg
3. Delete-at-end
4. Delete-at-pos
5. Display 2.

The linked list you have created is
20

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

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

Node * createNode (int value)
{
    Node *newnode = (Node*) malloc (sizeof(Node));
    newNode -> data = value;
    newNode -> next = NULL;
    return newNode;
}
void displaylist (Node *head)
{
    while (head != NULL)
    {
        pf ("%d ->", head -> data)
        head = head -> next
    }
    pf ("NULL\n");
}

Node * sortList (Node *head)
{
    if (head == NULL || head -> next == NULL)
    {
        return head;
```

```
int swapped;
Node *temp;
Node * end = NULL;

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

return head;
}

Node *reverselist (Node * head)
{
Node *prev = NULL;
Node *current = head;
Node * nextNode = NULL;
while (current != NULL)
{
nextNode = current -> next;
current -> next = prev
```

(Node)

```
        prev = current;
        current = nextNode;
    }
return prev;
}

Node * concatenate (Node *list1, Node
                                * list2)
{
    if (list1 == NULL)
            return list2;
    Node * temp = list1;
    while (temp -> next != NULL)
    {
    temp = temp -> next;
    }
     temp -> next = list2;
    return list1;
}
int main ()
{
   Node *list1 = createNode (3);
   list1 -> next = createNode (1);
   list1 -> next -> next = createNode (4);
   Node * list2 = createNode (2);
   list2 -> next = createNode (5);
    list2 -> next -> next = CreateNode (8);

   pf (" Original list 1: ");
   displaylist (list1);

   pf (" Original list 2:" );
   displaylist (list2);
```

```
list1 = sortlist(list1)
printf("sorted list1 :");
display List (List 1);

list1 = reverse List (list1);
printf ("Reversed List 1 :");
displayList (list 1);

Node * Concate nated = concatenated Lists (List1,
                                           List2);

printf (" Concatenated List: ");
displaylist (Concatenated);
return 0;
}
```

Output

Original list 1: 3 → 1 → 4 → NULL
Original list 2: 2 → 5 → NULL
Sorted list 1: 1 → 3 → 4 → NULL
Reversed list 1: 4 → 3 → 1 → NULL
Concatenated list: 4 → 3 → 1 → 2 → 5 → NULL

Implementation of Stack:

```
#include <stdio.h>
#include <stdlib.h>
#define pf printf
#define sf scanf

struct node {
    int data;
    struct node *next;
};
struct node *head, *newnode, *temp, *temp1;

void push()
{
  newnode = (struct node*) malloc(sizeof
                                  (struct node));
  pf("Enter the data to be pushed into the
     stack:");
  sf("%d", &newnode->data);
  newnode->next = head;
  head = newnode;
}

void display()
{
  pf("\n The stack you have created
      is :\n");
  while (temp1 != NULL)
  {
    pf("%d\n", temp->data);
    temp = temp->next;
  }
}
```

NULL

```
void pop()
{
    if (head == NULL)
        pf("\nThe stack is empty");
    else
    {
        temp = head;
        head = temp -> next;
        free(temp);
    }
}

int main()
{
    int ch;
    while (1)
    {
        pf("\n1.Push \n 2. Pop \n 3. Display
        \n 4. Exit Menu \n Choice : ");
        sf("%d", &ch);

    switch (ch)
    {
    case 1: push();
            display();
            break;
    case 2: pop();
            display();
            break;
    case 3: display();
            break;
    case 4: return 0;
    default :
            pf("\nInvalid input -- try
```

again \n\n\n");
}
}

Output :

1. Push
2. Pop
3. Display
4. Exit Menu
Choice : 1.
Enter the data to be pushed into the stack : 20

The stack you have created is :
20

play

1. Push
2. Pop
3. Display
4. Exit Menu
Choice : 1.
Enter the data to be pushed into the stack : 25

The stack you have created is :
25
20

1. Push
2. Pop
3. Display
4. Exit Menu
Choice : 3

The stack you have created is:
25
20.

1. Push
2 Pop
3. Display
4. Exit Menu
Choice: 2.

The stack you have created is:
20.

Implementation of Queue

```
#include<stdio.h>
#include<stdlib.h>
#define pf printf
#define sf scanf

struct node {
        int data;
        struct node *next;
};
struct node *head, *newnode, *temp,
                *temp1;

void pushqueue()
{
  newnode=(struct node *) malloc (sizeof(
                struct node));
pf("Enter the data to be pushed into
        the queue :");
sf("%d", &newnode->data);
newnode->next=head;
head=newnode;
}
Void display()
{
  pf("\n The queue you have created
                is :\n");
  temp=head;
while(temp!=NULL)
{
  pf("\t%d", temp->data);
    temp= temp->next;
}

}
```

```
void popqueue()
{
    if (head == NULL)
        pf("\nThe queue is empty)
    else
    {
        temp = temp1 = head;
        while (temp->next != NULL)
        {
            temp1 = temp;
            temp = temp->next;
        }
        temp1->next = NULL;
        free(temp);
    }
}

int main()
{
    int ch;
    while (1)
    {
        pf("\n1. Push \n2. Pop \n3. Display \n
            4. Exit Menu \nChoice: ");
        sf("%d", &ch);

    switch (ch){
        case 1: pushqueue();
                display();
                break;
        Case 2: popqueue();
                display();
                break;
```

```
case 3: display();
        break;
case 4: return 0;
default:
        bf("\n Invalid input -- try again
                                \n\n\n");
    }
}
```

Output:

1. Push
2. Pop
3. Display
4. Exit Menu

Choice: 1

Enter the data to be pushed into the queue: 2

The queue you have created is:

2

1. Push
2. Pop
3. Display
4. Exit Menu

Choice: 1

Enter the data to be pushed into the queue: 4

The queue you have created is:

4 2

1. Push
2. Pop
3. Display
4. Exit Menu

Choice: 3

The queue you have created is:
4

1. Push
2. Pop
3. Display
4. Exit Menu
Choice: 3

The queue you have created is:
4

1. Push
2. Pop
3. Display
4. Exit Menu
Choice: 4

LAB

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

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

struct node * addtoempty(struct node * head, int data)
{
    struct node *temp=malloc(sizeof(struct node));
    temp->prev=NULL;
    temp->data=data;
    temp->next=NULL;
    head=temp;
    return head;
}

struct node * createList(struct node * head)
{
    int n, data, i;
    printf("Enter the number of node: ");
    scanf("%d",&n);

    if(n==0)
        return head;
    pf("\nEnter the element for the node 1:");
    scanf("%d",&data);
    head = addToEmpty(head, data);
```

```
for (i=01; i<n; i++)
    pf("Enter the element for the node
        %d: ", i+1);
    sf("%d", &data);
    head = addAtEnd(head, data);
}
return head;
}

struct node * addatend(struct node * head,
                       int data)
{
    struct node *tempnew, *tp;
    tempnew = NULL;
    tempnew = addtoempty(tempnew, data);

    tp = head;
    while(tp->next != NULL)
    {
        tp = tp->next;
    }
        tp->next = tempnew;
        tempnew->prev = tp;
        return head;
    }

int main()
{
    int choice;
    while (1) {
    pf("\n 1. Create a doubly linked list \n
        2. Insert a node before a node\n
```

3. Delete a node

```
struct node * delete_specific_value (struct node *head, int val)
{
    struct node * temp = head; struct node *tp;
    if (temp == NULL)
    {
        pf("Empty linked list");
    }
    else if (temp->next == NULL)
    {
        temp = head = NULL;
        pf("In The only node has been de
    }
    else {
        while (temp->next != NULL){
            if (temp->data == val){
                tp = temp;
                temp->prev->next = temp->next
                temp->next->prev = temp->prev;
                temp = hed;
            }
            temp = temp->next;
        }
    }
    return head;
}
```

```
struct node *insert_bef_pos (struct node *head,
                             int data, int pos)
{
    struct node *temp = head;
    struct node *t1, *t2;
    struct node *newp = NULL;
    newp = addtoempty (newp, data);
    if (pos == 1)
        head = addtoempty (head, data);
    else {
        while (pos != 1) {
            temp = temp -> next;
            if ( temp -> next == NULL) {
                pf("\n The number of node
                    is less than %d", pos);
            }
            pos--;
        }
        t1 = temp -> prev;
        t2 = temp -> next;
        t1 -> next = newp;
        t2 -> prev = newp;
        newp -> next = t2;
        newp -> prev = t1;
    }
    return head;
}

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

```
}
}

int main()
{ int choice, int data, int pos;
  struct node * head = NULL;
  while (1) {
    pf("\n Enter your choice \n
        1. create \n 2. insert at beg of pos \n
        3. delete specific value \n 4. display \n
        choice : ");
    sf("%d", &ch);
    switch (choice) {
    case 1 : head = createList (head);
             display(head);
             break;
    case 2: pf("\n Enter the data & position:");
     sf("%d %d", &data, &pos);
     head = insert_bef_pos(head, data, pos);
        display(head);
        break;
    case 3: pf("\n Enter the value of node
                to be deleted");
            sf("%d", &data);
            head = delete_specific_value(head,
                                         data);

            display(head);
            break;
    Case 4 : display(head);
    }
  }
}
```

Output:

Enter your choice
1. Create
2. Insert at beg of pos
3. delete specific value
4. display
choice : 1

Enter the number of node : 3
Enter the element of the node 1 : 6
Enter the element of the node 2 : 8
Enter the element of the node 3 : 10

6 → 8 → 10

Enter your choice :
1. Create
2. Insert at beg of pos
3. delete specific value
4. display
choice : 2

Enter the data & position : 0 2

6 → 0 → 8 → 10.

Enter your choice
1. Create.
2. Insert at beg of pos
3. delete specific value
4. display
Choice : 3.

Enter the value of value of the node to be deleted : 0

6 → 8 → 10

Sneha
1/2/24

Leet Code: Split Linked List in Parts

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

int getLength(struct ListNode * head){
    int length = 0;
    while (head != NULL){
        length++;
        head = head->next;
    }
    return length;
}

struct ListNode ** splitListToParts
(struct ListNode * head, int k, int *
                        returnSize){
    int length = getLength(head);
    int partSize = length / k;
    int remainder = length % k;

struct ListNode ** result = (struct ListNode*)
        malloc(k * sizeof(struct ListNode));
*returnSize = k;

for(int i=0; i<k; i++){
    int currentPartSize = partSize +
                    (i < remainder ? 1 : 0);

if (currentPartSize == 0){
        result[i] = NULL; }
```

*return
return
}

o/p:
[[1] [2

*return size = k;
return result;
}

o/p:
[[1] [2] [3] [5] [6] [] [] []]

LAB - 8

```
struct node
{
    int data;
    struct node *left, *right;
};

struct node* create ( int data)
{
    struct node *newnode;
    newnode= (struct node*) malloc
        (sizeof(struct node));
    newnode -> data= data;
    newnode -> left = node -> right = NULL;
    return node;
}

struct node* insert (struct node* root,
int data) {
if (root == NULL) return newnode(data);

if (data <= root->data)
    root -> left = insert( root -> left , data);
else
    root->right= insert (root -> right , data);

return root;
}
```

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

void preorderTraversal (struct node * root) {
    if (root != NULL) {
        printf ("%d \t", root->data);
        preorderTraversal (root->left);
        preorderTraversal (root->right);
    }
}

void postorderTraversal (struct node *root) {
    if (root != NULL) {
    postorderTraversal (root->left);
        ~~print~~
    postorderTraversal (root->right);
    printf ("%d \t", root->data);
    }
}

int main () {
    struct node * root = NULL ; int data, choice;
do {
    printf ("Enter your choice : \n 1. Insert
\n 2. Print Inorder \n 3. Print Preorder \n 4. Print
Post Order \n. 5. Exit \n");
    scanf ("%d", &choice);
    switch (choice) {
        case 1 : printf ("Enter the value to be
            inserted :");
```

```
        scanf ("%d", &data);
        root = insert (root, data);
        break;
    case 2: printf ("\nInoder traversal of
        binary tree is \n");
        inorder (root);
        printf ("\n");
        break;
    case 3: printf ("\n Preorder traversal of
        binary tree is :\n");
        preorderTraversal (root);
        printf ("\n");
        break;
    case 4: printf ("\n Postorder traversal of
        binary tree is :\n");
        postorderTraversal (root);
        printf ("\n");
        break;
    case 5: printf ("Exiting ....\n");
        break;
    default: printf ("Invalid choice. Please
            try again.\n");
    }
} while (choice != 5);
    return 0;
}
```

Enter your choice :
1. Insert
2. Print Inorder
3. Print Preorder
4. Print Postorder
5. Exit.

1.
Enter the value to be inserted :
7.

Enter your choice :
1. Insert
2. Print Inorder
3. Print Preorder
4. Print Postorder
5. Exit.

1.
Enter the value to be inserted :
6

Enter your choice :
1. Insert
2. Print Inorder
3. Print Preorder
4. Print Postorder
5. Exit.

1.
Enter the value to be inserted :
8

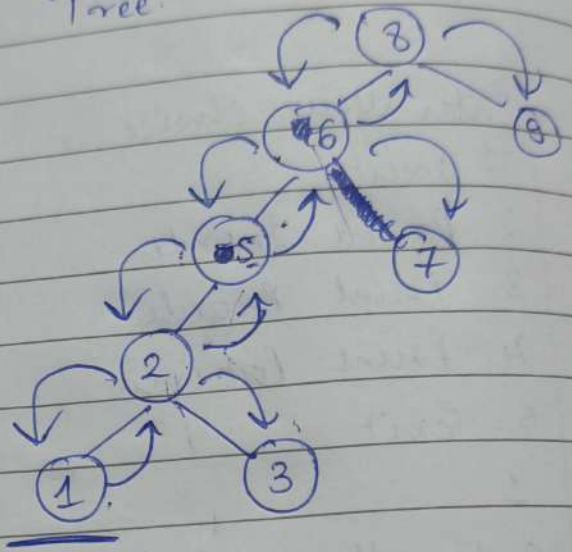
Enter your choice
1. Insert

3. Print Preorder
4. Print Post order
5. Exit

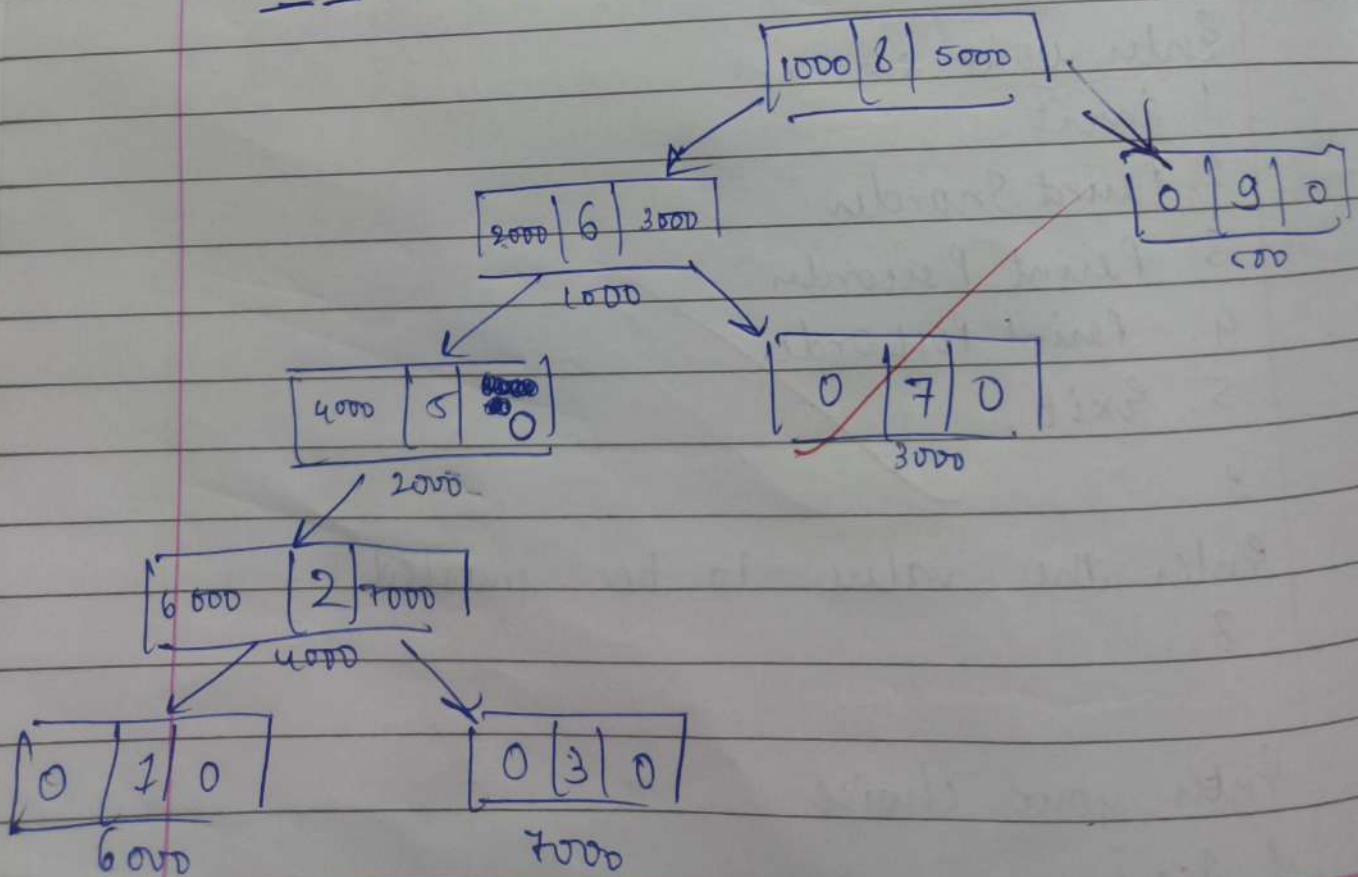
2.
Preorder traversal of binary tree is:
6 7 8.

Tracing Preorder Eg: Tree

6 7 8



1 2 3 5 6 7 8 9



Leet code

Rotation of Linked Linked k times.

```
int GretLength (struct ListNode * head)
{
    struct ListNode *temp = head;
    int count = 0;
    while (temp != NULL){
        count ++
        temp = temp->next;
    }
    return count;
}

struct Listnode * riotateRight (struct Listnode
* head, int k ){
    int length = GgetLength ( head);
    struct Listnode *temp = head ;
    struct Listnode *temp1 ;

If (head == NULL) return head;
else if (head->next == NULL) return head;
else {
    int i = length % k;
    while (i>0) {
        temp = head;
        while (temp->next = NULL
        {
            temp1 = temp;
            temp = temp->next;
        }
        temp1->next = temp->next;
        temp->next = head;
        head = temp;
    }
}
```

Output:
Test Case 1:
[1, 2, 3, 4, 5] , k = 6
↳ [5, 1, 2, 3, 4]

Test Case 2:
head = [0, 1, 2]
k = 4:

Output: [2, 0, 1]

Sneha
2/3/24

Lab-9 (BSF)

```
#include <stdio.h>
int a[20][20], q[20], visited[20], n, i, j, f=0, r=-1;

void bfs(int v)
{
  for(i=1; i<=n; i++)
    if (a[v][i] && !visited[i])
      q[++r] = i;
  if (f<=r)
  {
    visited[q[f]] = 1;
    bfs(q[f++]);
  }
}

int main()
{
  int v;
  printf("\n Enter the number of vertices:");
  scanf("%d", &n);

  for(i=1; i<=n; i++)
  {
    q[i] = 0;
    visited[i] = 0;
  }

printf("Enter graph data in matrix form:\n");
for(i=1; i<n; i++)
    for(j=1; j<=n; j++)
        scanf("%d", &a[i][j]);
```

Output:

Enter the
Enter the
0
1
1
1
1
0

Enter t
The no
1

```
printf("\n Enter the starting vertex:");
scanf("%d",&v);
bfs(v);
printf("The nodes which are reachable are:\n");
for(i=1; i<=n; i++)
    if(visited[i])
        printf("%d\t", i);

return 0;
}
```

Output:

Enter the number of vertices: 6
Enter the graph data in matrix form:

```
0 1 1 1 1 0
1 0 0 1 0 0
1 0 0 1 1 0
1 1 1 0 0 0
1 0 1 0 0 0
0 0 0 0 0 0
```

Enter the starting vertex: 4
The node which are reachable are:
1 2 3 4 5

Lab 9 Using DFS

```
#include <stdio.h>
#include <conio.h>
int a[20][20], s[20], n;
void dfs(int v)
{
    int i;
    s[v]=1;
    for(i=1; i<=n; i++)
        if(a[v][i] && !s[i])
        {
            printf("\n%d->%d", v, i);
            dfs(i);
        }
}

int main()
{
    int i, j, count=0;
    printf("\nEnter number of vertices:");
    scanf("%d", &n);
    for(i=1; i<=n; i++)
    {
        s[i]=0;
        for(j=1; j<=n; j++)
            a[i][j]=0;
    }
    printf("Enter the adjacency matrix:\n");
    for(i=1; i<=n; i++)
        for(j=1; j<=n; j++)
            scanf("%d", &a[i][j]);
    dfs(1);
    printf("\n");
```

```
for(i=1; i<
    if(
}
if(count
    print
else
    print
return
}.

Enter the
Enter the
0 1
1 0
1 0
1 1
1 0
0 0

1->2
2->4
4->
3->5
3->
Graph
```

```
for(i=1; i<=n; i++)
  {
    if(s[i])
      count++;
  }
if(count==n)
  printf("Graph is connected");
else
  printf("Graph is not connected");
return 0;
}.
```

Enter the number of vertices : 6
Enter the adjacency matrix :

```
0 1 1 1 1 0
1 0 0 1 0 0
1 0 0 1 1 1
1 1 1 0 0 1
1 0 1 0 0 0
0 0 1 1 0 0
```

1 → 2
2 → 4
4 → 3
3 → 5
3 → 6

Graph is connected

22/02/24 HackerRank

Swapping Subtrees function:

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

int** swapNodes(int indexes_rows, int indexes_columns, int** indexes, int queries_count, int* queries, int* result_rows, int* result_columns){

    Node **nodes = (Node**) malloc((indexes_rows+1) * sizeof(Node*));

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

        int leftIndex = indexes[i][0];
        int rightIndex = indexes[i][1];
        if(leftIndex != -1) nodes[i+1]->left = nodes[leftIndex];

        if(rightIndex != -1) nodes[i+1]->right = nodes[rightIndex];
    }
```

Output:

Input:

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

Expected

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

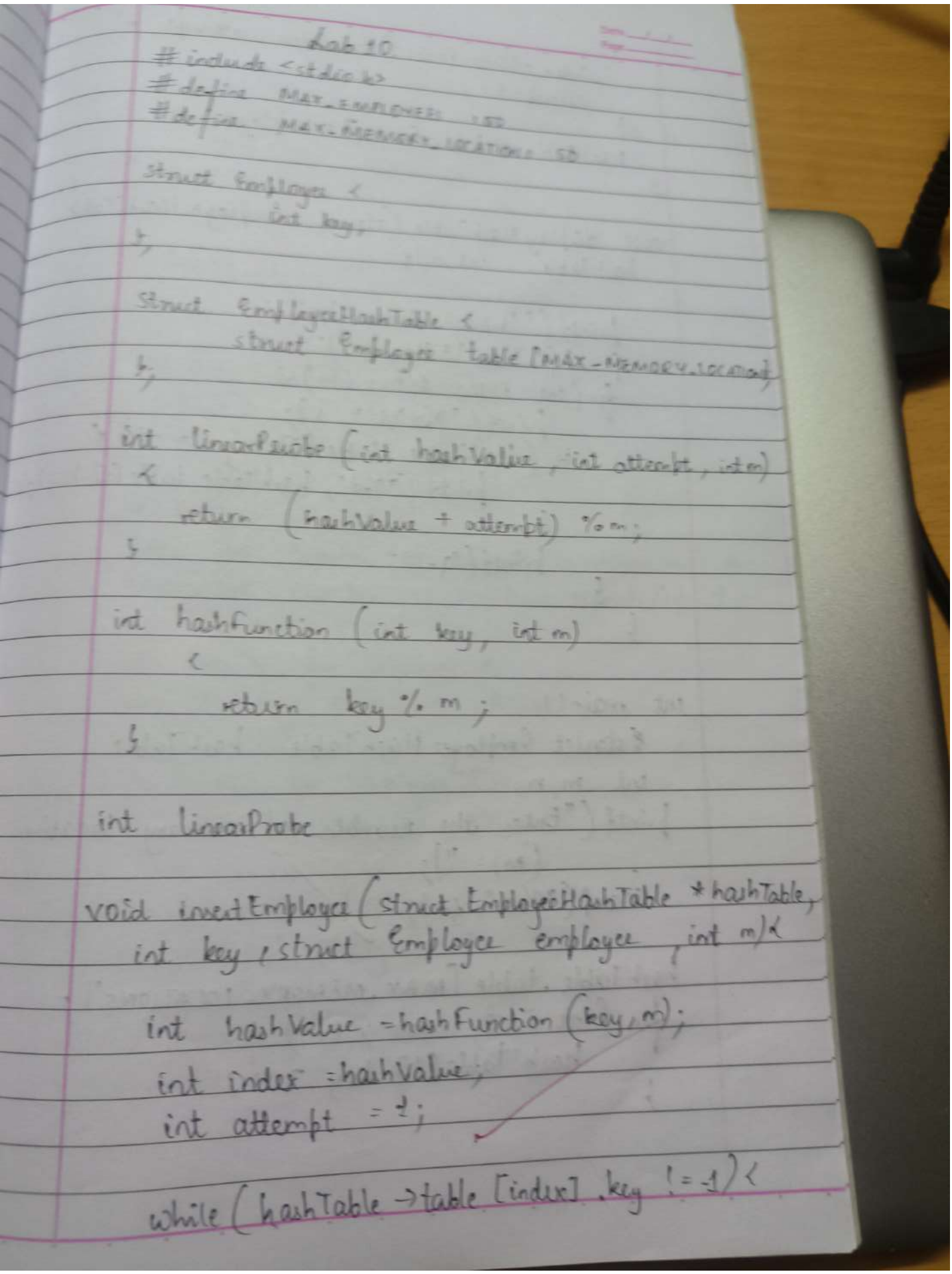
Output:

Input:

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

Expected Output:

3 1 2
2 1 3



Lab 10

```
#include <stdio.h>
#define MAX_EMPLOYEES 100
#define MAX_MEMORY_LOCATIONS 50

struct Employee {
    int key;
};

struct EmployeeHashTable {
    struct Employee table[MAX_MEMORY_LOCATIONS];
};

int linearProbe(int hashValue, int attempt, int m)
{
    return (hashValue + attempt) % m;
}

int hashFunction(int key, int m)
{
    return key % m;
}

int linearProbe

void insertEmployee(struct EmployeeHashTable *hashTable,
  int key, struct Employee employee, int m){

    int hashValue = hashFunction(key, m);
    int index = hashValue;
    int attempt = 1;

    while(hashTable->table[index].key != -1){
```

```
        index = linearProbe(hashValue, attempt, m);
        attempt++;
    }
    hashTable->table[index] = employee;
}

void displayHashTable(struct EmployeeHashTable
    hashTable, int m){

    printf("\nHash Table :\n");
    printf("Index\t Key\n");
    for(int i=0; i<m; i++){
        printf("%d\t", i);
        if(hashTable.table[i].key != -1){
            printf("%d", hashTable.table[i].key);
        }
        printf("\n");
    }
}

int main()
{
    struct EmployeeHashTable hashTable;
    int m, n;
    printf("Enter the number of memory location
            (m): ");
    scanf("%d", &n);

    hashTable.table[MAX_MEMORY_LOCATIONS];
    for(int i=0; i<m; i++){
        hashTable[i].key = -1;
    }
```

```
for (int i=0; i<n; i++) {
        struct Employee employee;
    printf("\nEnter details of Employee %d:\n", i+1);
    printf("Enter the key (4 digits): ");
    scanf("%d", &employee.key);
/*  printf("Enter the Name:"); */

    insertEmployee(&hashTable, employee.key, employee,
                                                m);
}
    displayHashTable(hashTable, m);
    return 0;
}
```

Output:

```
Enter the number of memory locations (m): 4
Enter the number of employee records (n): 4

Enter details for Employee 1:
Enter the key 4 (digit): 2034

Enter details for Employee 2:
Enter the key (4digit): 1024

Enter details for Employee 3:
Enter the key (4digit): 3011

Enter details for Employee 4:
Enter the key (4 digit): 3033
```

P.T:0.

Hash Table

Index	Key
0	1024
1	3033
2	2034
3	3011

Sneha
2/3/24