VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

TISSA MARIA (1BM22CS309)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
Dec 2023- March 2024

B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering

This is to certify that the Lab work entitled **"DATA STRUCTURES"** carried out by TISSA MARIA **(1BM22CS309)**, who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2023-24. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab - **(23CS3PCDST)**work prescribed for the said degree.

Prof. Lakshmi Neelima
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Course outcomes:

CO1	Apply the concept of linear and nonlinear data structures.
CO2	Analyze data structure operations for a given problem
CO3	Design and develop solutions using the operations of linear and nonlinear data structure for a given specification.
CO4	Conduct practical experiments for demonstrating the operations of different data structures.

Lab program 1:

Write a program to simulate the working of stack using an array with the following:
a) Push
b) Pop
c) Display
The program should print appropriate messages for stack overflow, stack underflow.

```
#include <stdio.h>
#define SIZE 5
int arr[5], top=-1;
void push(int value){
if(top==SIZE-1){
    printf("\nThe stack is filled\n");
```

```
}else{

    top+=1;
    arr[top]=value;
    printf("\nSuccessfully Added\n");}
   }
void pop(){
    if(top==-1){
    printf("\nThe stack is empty\n");
}else{
    printf("\n%d has been deleted from the array\n",arr[top]);
    top=top-1;
   }
}
void display(){
    if(top==-1){
    printf("\nThe stack is empty\n");
}else{
    for(int i=0;i<=top;i++){
    printf("%d\n",arr[i]);}

   }

}
int main(){
    int boolean=1,choice,value;
    while(boolean){
        printf("\n1.PUSH\t2.PULL\t3.DISPLAY\t4.EXIT\n");
        scanf("%d",&choice);
        switch(choice){
        case 1:{printf("Enter a value");
            scanf("%d",&value);
                push(value);
                break;}
        case 2:pop();
        break;
        case 3:display();
        break;
        case 4:boolean=0;
        break;
        default:printf("Invalid Input");
        }
    }
    return 0;
    }
```

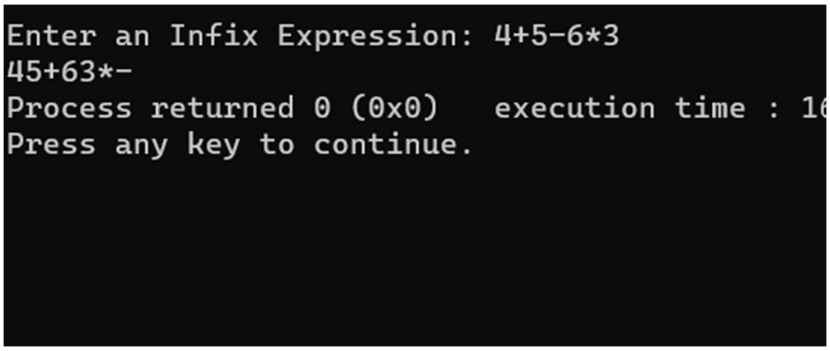
```
1.PUSH  2.PULL  3.DISPLAY       4.EXIT
1
Enter a value2

Successfully Added

1.PUSH  2.PULL  3.DISPLAY       4.EXIT
1
Enter a value4

Successfully Added

1.PUSH  2.PULL  3.DISPLAY       4.EXIT
1
Enter a value5

Successfully Added

1.PUSH  2.PULL  3.DISPLAY       4.EXIT
2

5 has been deleted from the array

1.PUSH  2.PULL  3.DISPLAY       4.EXIT
3
2
4

1.PUSH  2.PULL  3.DISPLAY       4.EXIT
```

Infix to Postfix

```c
#include <stdio.h>
#include <string.h>

#define MAX 100

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

void push(char symbol);
char pop();
int isEmpty();
int precedence(char symbol);
void print();
void inToPost();
```

```
int main() {
    printf("Enter an Infix Expression: ");

    gets(infix);
    inToPost();
    print();
    return 0;
}

void inToPost() {
    int i = 0, j = 0;
    char symbol, next;

    for (; i < strlen(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() && precedence(stack[top]) >=
precedence(symbol)) {
                    postfix[j++] = pop();
                }
                push(symbol);
                break;
            default:
                postfix[j++] = symbol;
        }
    }

    while (!isEmpty()) {
        postfix[j++] = pop();
    }

    postfix[j] = '\0';
}
```

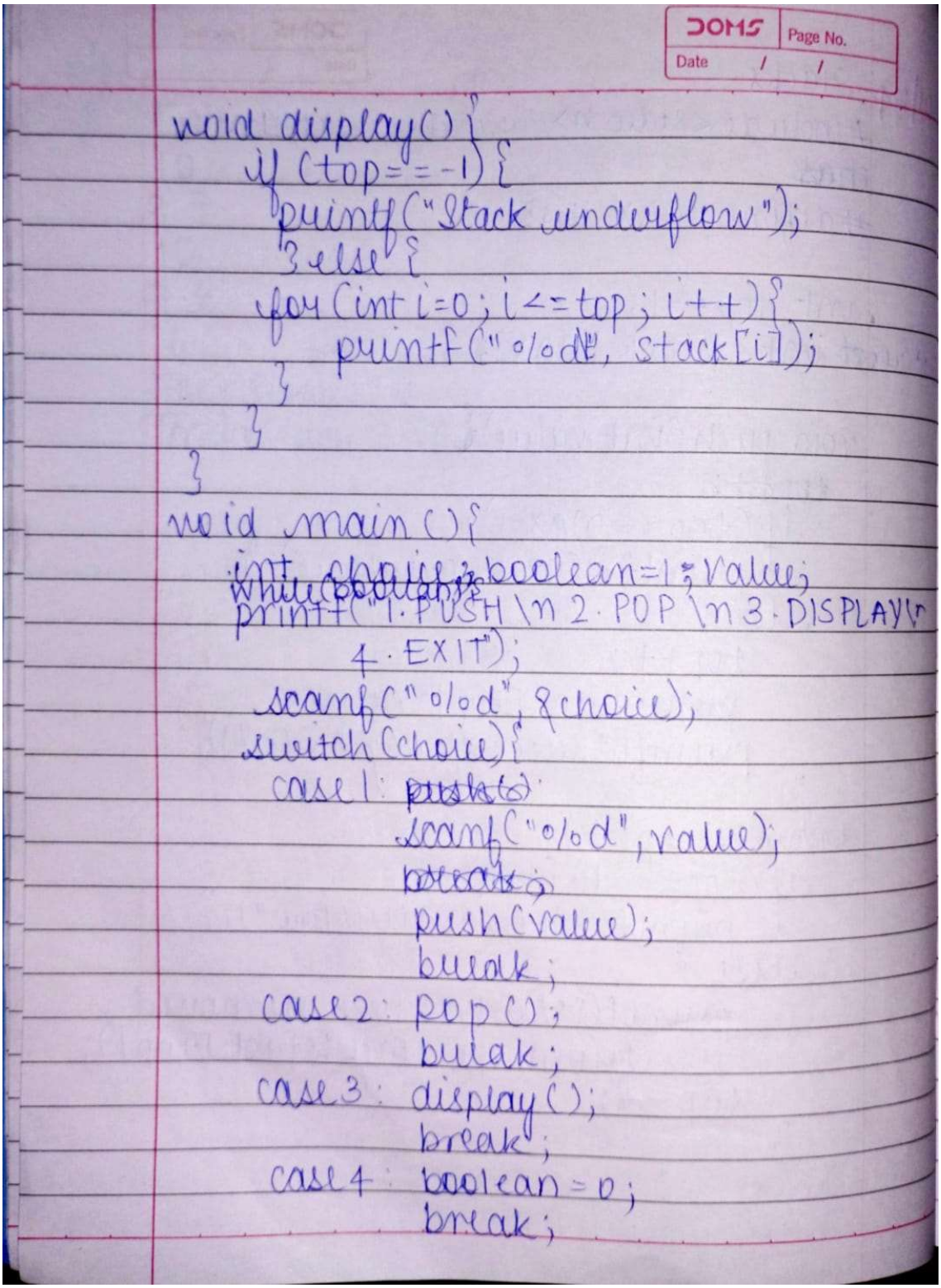
```
void print() {
    for (int i = 0; i < strlen(postfix); i++) {
        printf("%c", postfix[i]);
    }

}

void push(char symbol) {
    top++;
    stack[top] = symbol;
}

char pop() {
    char next = stack[top];
    top--;
    return next;
}

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

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


Enter an Infix Expression: 4+5-6*3
45+63*-
Process returned 0 (0x0)   execution time : 1
Press any key to continue.

DOMS | Page No.
Date / /

8/01/24 · Stacks

```
#include <stdio.h>

#define MAX 5

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

void push(int value){
    if (top == MAX-1){
        printf("Stack overflow");}
    else{
        top++;
        stack[top]=value;
        printf("Successfully Added");
    }
void pop(){
    if (top==-1){
        printf("Stack underflow");}
    else{
        printf("%od has been removed
            from the stack",stack[top]);
        top--;
    }
}
```



```
void display(){
    if (top==-1){
        printf("Stack underflow");
    }else{
        for(int i=0; i<=top; i++){
            printf("%d", stack[i]);
        }
    }
}
void main(){
    int choice, boolean=1, value;
    while(boolean){
        printf("1.PUSH\n2.POP\n3.DISPLAY\n
            4.EXIT");
        scanf("%d", &choice);
        switch(choice){
            case 1: push();
                    scanf("%d", value);
                    push(value);
                    break;
            case 2: pop();
                    break;
            case 3: display();
                    break;
            case 4: boolean=0;
                    break;
```

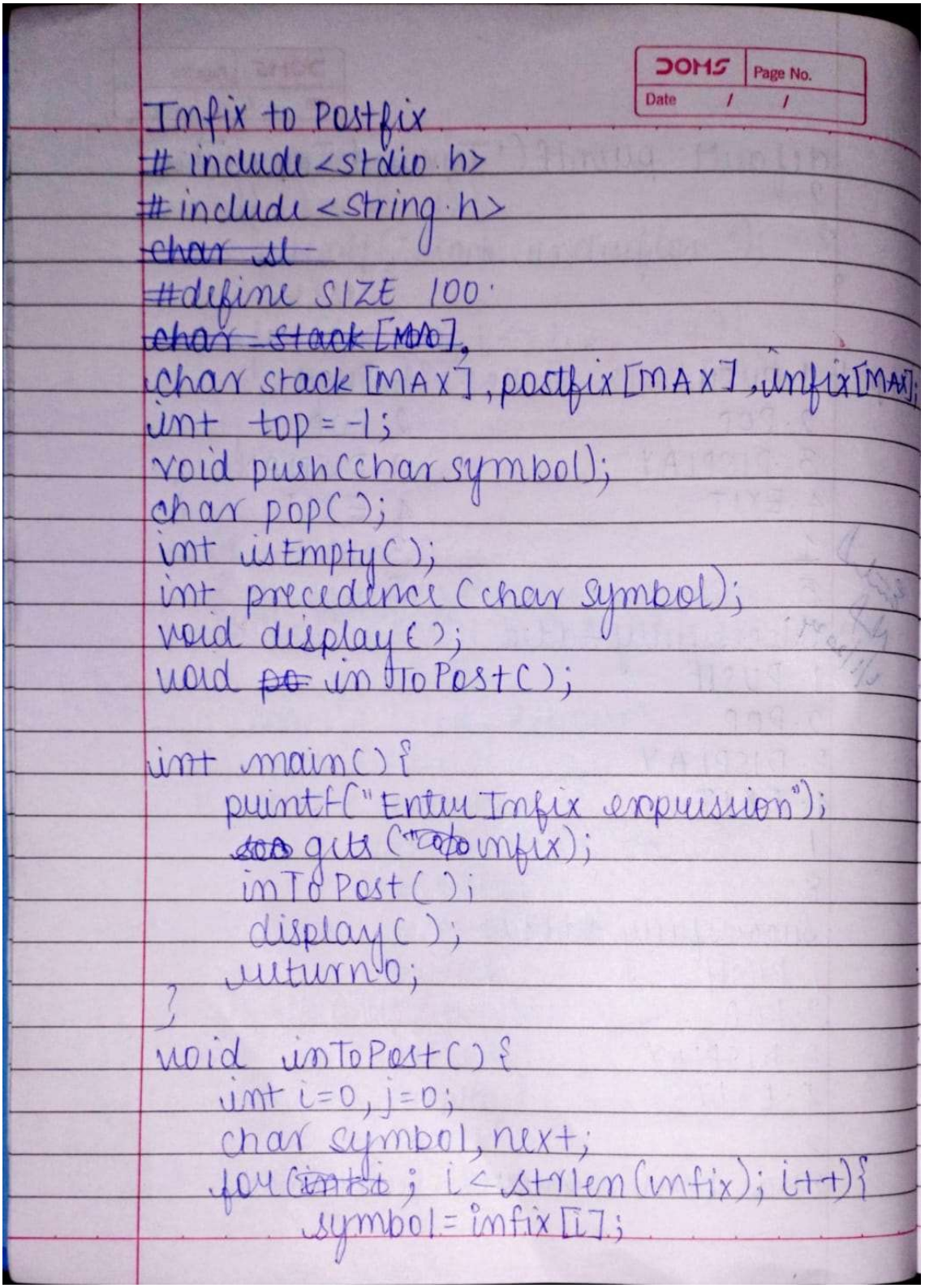
```
default: printf("Invalid Input");
}
}
}
```

output

```
1. PUSH
2. POP
3. DISPLAY
4. EXIT
1
5
Successfully Added
1. PUSH
2. POP
3. DISPLAY
4. EXIT
1
8
Successfully Added
1. PUSH
2. POP
3. DISPLAY
4. EXIT
2
8 has been removed from the stack
```

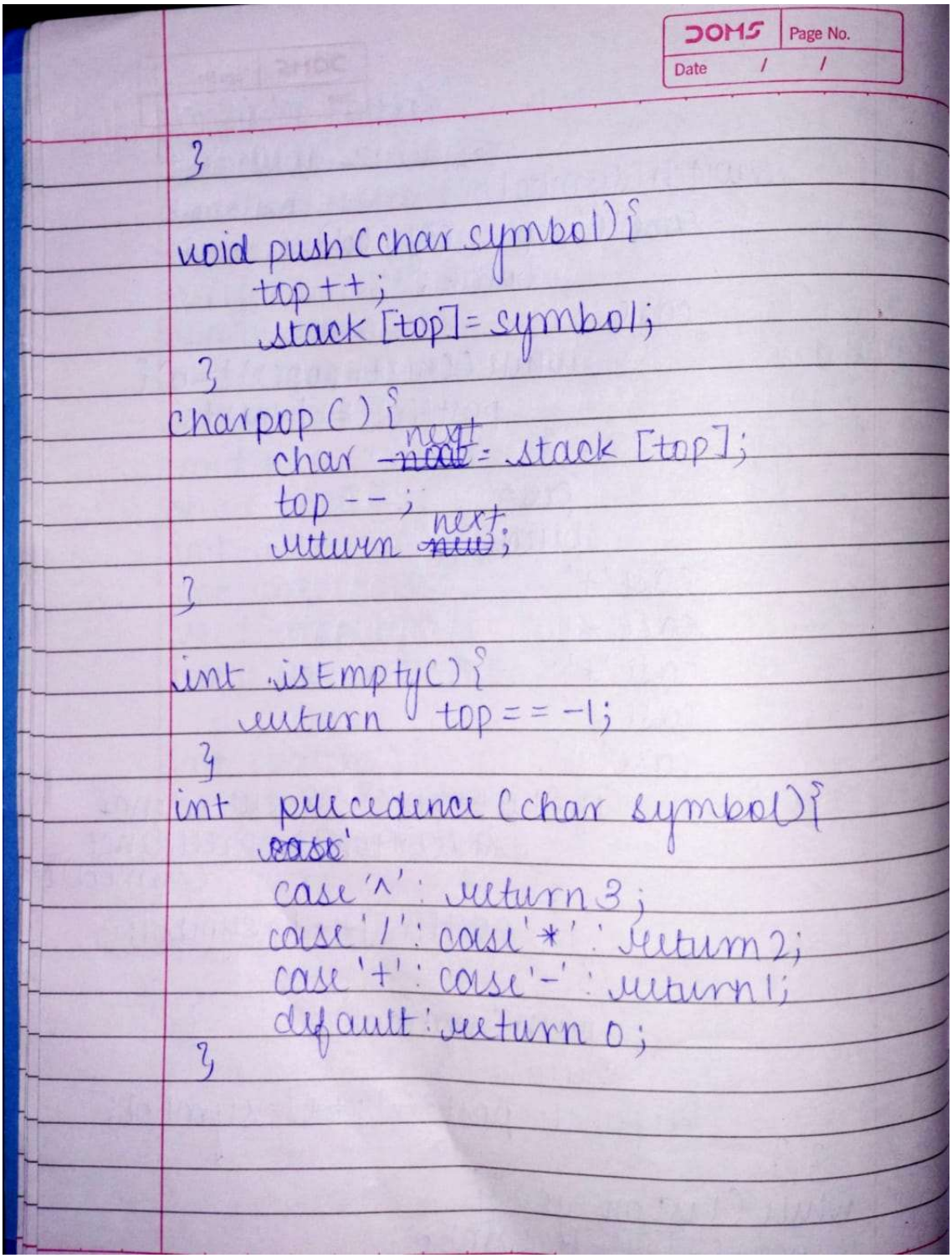
```
1. PUSH
2. POP
3. DISPLAY
4. EXIT
4
3
5
```

Evaluated 6/1/24



Infix to Postfix

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

int main(){
    printf("Enter Infix expression");
    gets(infix);
    inToPost();
    display();
    return 0;
}
void inToPost(){
    int i=0, j=0;
    char symbol, next;
    for(i; i<strlen(infix); i++){
        symbol = infix[i];
```



```
}

void push(char symbol){
    top++;
    stack[top] = symbol;
}
charpop(){
    char next = stack[top];
    top--;
    return next;
}

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

```
switch(symbol){
    case '(' : push(symbol);
               break;
    case ')':
            while((next=pop())!=0){
                postfix[j++]=next;
            }
            break;
    case '+':
    case '-':
    case '*':
    case '/':
    case '^':
        if(!isEmpty() && precedence
            (stack[top])>=precedence
                                (symbol)){
            postfix[j++]=symbol;
        }
        push(symbol);
        break;
    default: postfix[j++]=symbol;
}}
while(!isEmpty()){
    postfix[j++]=symbol;
}postfix[j]='\0';
```

Lab 2 : Queue

```
 #include <stdio.h>

#define MAX 5
int front=-1, rear=-1, q[MAX];
void enqueue(int value){
    if(front==-1 && rear ==-1){
        front=0;
        rear=0;
        q[rear]=value;
    }else if(rear==MAX-1){
        printf("Overflow");}
    else{
        rear++;
        q[rear]=value;
    }
    }
void dequeue() {
    if (front == -1) {
        printf("Underflow");
    } else {
        if (front > rear) {
            front = -1;
            rear = -1;
        } else {
            printf("%d deleted", q[front]);
            front++;
        }
    }
}
void display(){
if(front==-1){
    printf("Underflow");
}else{
for(int i=front;i<=rear;i++){
    printf("%d",q[i]);
}

}}

int main(){
    int boolean=1,choice,value;
    while(boolean){
        printf("\n1.Enqueue\t2.Dequeue\t3.DISPLAY\t4.EXIT\n");
        scanf("%d",&choice);
        switch(choice){
        case 1:{printf("Enter a value");
            scanf("%d",&value);
```

```
                enqueue(value);
                break;}
        case 2:dequeue();
        break;
        case 3:display();
        break;
        case 4:boolean=0;
        break;
        default:printf("Invalid Input");
        }
    }
    return 0;
    }
```

```
1.Enqueue       2.Dequeue       3.DISPLAY       4.EXIT
1
Enter a value2

1.Enqueue       2.Dequeue       3.DISPLAY       4.EXIT
1
Enter a value3

1.Enqueue       2.Dequeue       3.DISPLAY       4.EXIT
1
Enter a value4

1.Enqueue       2.Dequeue       3.DISPLAY       4.EXIT
1
Enter a value5

1.Enqueue       2.Dequeue       3.DISPLAY       4.EXIT
2
2 deleted
1.Enqueue       2.Dequeue       3.DISPLAY       4.EXIT
3
345
1.Enqueue       2.Dequeue       3.DISPLAY       4.EXIT
```

Lab - III

8/01/24 Queue

```
#include <stdio.h>
#define MAX 5
int front = -1, rear = -1, q[MAX];
void enqueue(int value){
    if(front == -1 && rear == -1){
        printf("Overflow"); front = rear = 0;
    }else if(rear == MAX-1){
    q[rear] = value;
    }else if(rear = MAX-1){
        printf("Overflow");
    }else{
    q[++rear] = value;}}
void dequeue(){
if(front == -1){
    printf("Underflow");
}else{
    if(front > rear){
        front = -1;
    }else{
        printf("%d", q[front]);
        front++;
    }
}
```

```
void display(){
   if(front == -1){
      printf("Underflow");
}else{
   for(int i=front; i<=rear; i++){
        printf("%d", q[i]);
}}

int main(){
   int boolean=1, choice, value;
   while(boolean){
        printf("1.Enqueue\n2.Dequeue\n3.Display
              \n4.Exit\n");
        scanf("%d",&choice);
        switch(choice){
        case 1: printf("Enter a value");
              scanf("%d", value);
              enqueue(value);
              break;
        case 2: dequeue(); break;
        case 3: display; break;
        case 4: boolean = 0; break;
        default: printf("Invalid Input"); break;
   }return 0;
}
```

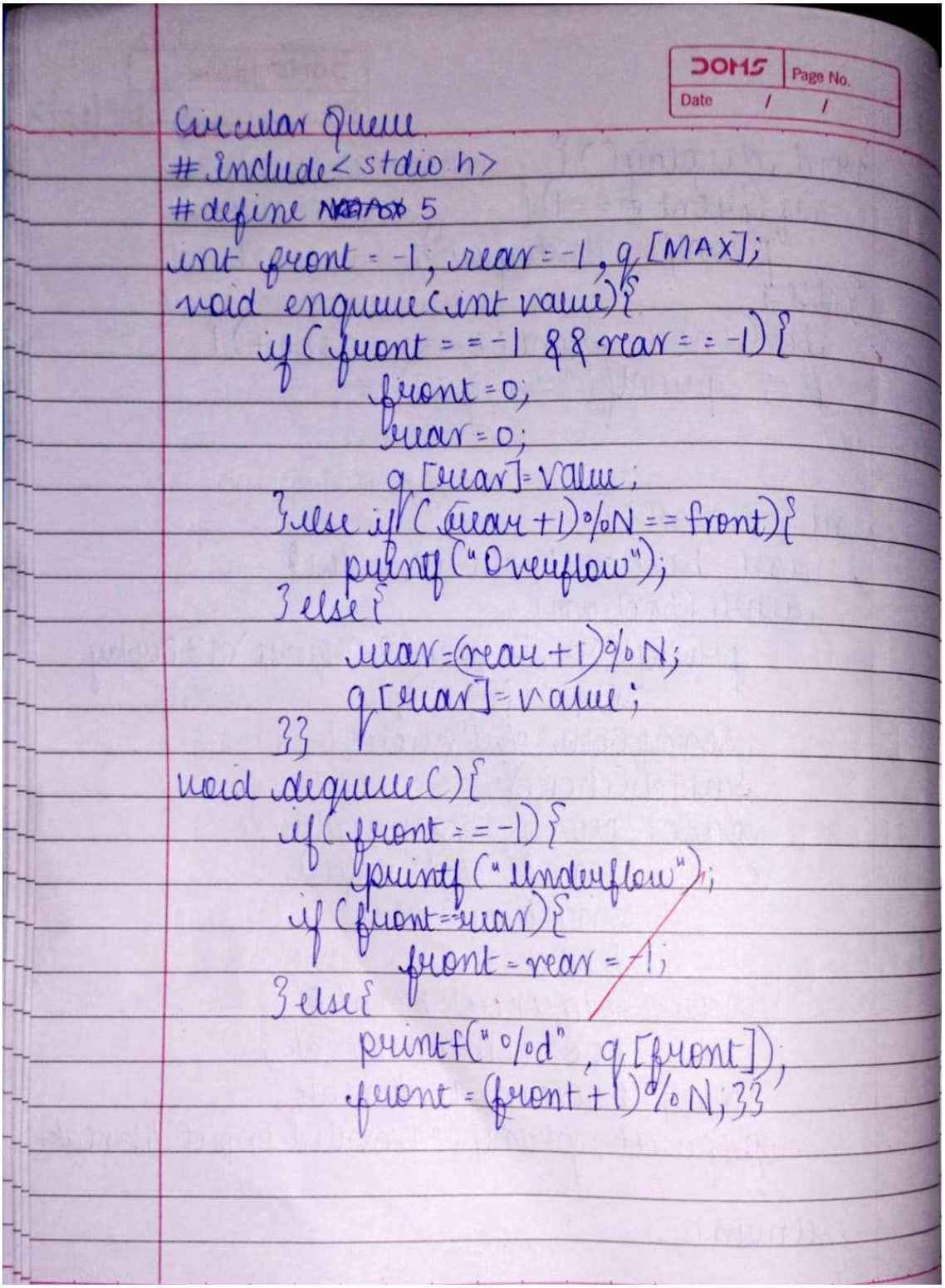
Circular Queue

```
#include <stdio.h>
#define MAX 5
int front=-1, rear=-1, q[MAX];
void enqueue(int value){
    if(front==-1 && rear ==-1){
        front=0;
        rear=0;
        q[rear]=value;
    }else if((rear+1)%MAX==front){
        printf("Overflow");}
    else{
        rear=(rear+1)%MAX;
        q[rear]=value;
    }
    }
void dequeue(){
if(front==-1){
    printf("Underflow");

}else{
    if(front==rear){
        front=-1;
        rear=-1;
    }else{
    printf("%d deleted",q[front]);
    front=(front+1)%MAX;}
    }
    }
void display(){
if(front==-1){
    printf("Underflow");
}else{
    int i=front;
while(i!=rear){
    printf("%d",q[i]);
    i=(i+1)%MAX;
}

}}

int main(){
    int boolean=1,choice,value;
    while(boolean){
        printf("\n1.Enqueue\n2.Dequeue\n3.DISPLAY\n4.EXIT\n");
        scanf("%d",&choice);
        switch(choice){
        case 1:{printf("Enter a value");
            scanf("%d",&value);
                enqueue(value);
                break;}
        case 2:dequeue();
        break;
```

```
        case 3:display();
        break;
        case 4:boolean=0;
        break;
        default:printf("Invalid Input");
        }
    }
    return 0;
    }
```

```
1.Enqueue
2.Dequeue
3.DISPLAY
4.EXIT
1
Enter a value2

1.Enqueue
2.Dequeue
3.DISPLAY
4.EXIT
1
Enter a value4

1.Enqueue
2.Dequeue
3.DISPLAY
4.EXIT
1
Enter a value5

1.Enqueue
2.Dequeue
3.DISPLAY
4.EXIT
2
2 deleted
1.Enqueue
2.Dequeue
```

```
2.Dequeue
3.DISPLAY
4.EXIT
1
Enter a value4

1.Enqueue
2.Dequeue
3.DISPLAY
4.EXIT
1
Enter a value5

1.Enqueue
2.Dequeue
3.DISPLAY
4.EXIT
2
2 deleted
1.Enqueue
2.Dequeue
3.DISPLAY
4.EXIT
3
4
1.Enqueue
2.Dequeue
3.DISPLAY
4.EXIT
```



Circular Queue

```
#include<stdio.h>
#define MAX 5
int front = -1, rear = -1, q[MAX];
void enqueue(int value){
    if(front == -1 && rear == -1){
        front = 0;
        rear = 0;
        q[rear] = value;
    }else if((rear+1)%N == front){
        printf("Overflow");
    }else{
        rear = (rear+1)%N;
        q[rear] = value;
    }}
void dequeue(){
    if(front == -1){
        printf("Underflow");
    if(front == rear){
        front = rear = -1;
    }else{
        printf("%d", q[front]);
        front = (front+1)%N;}}
```

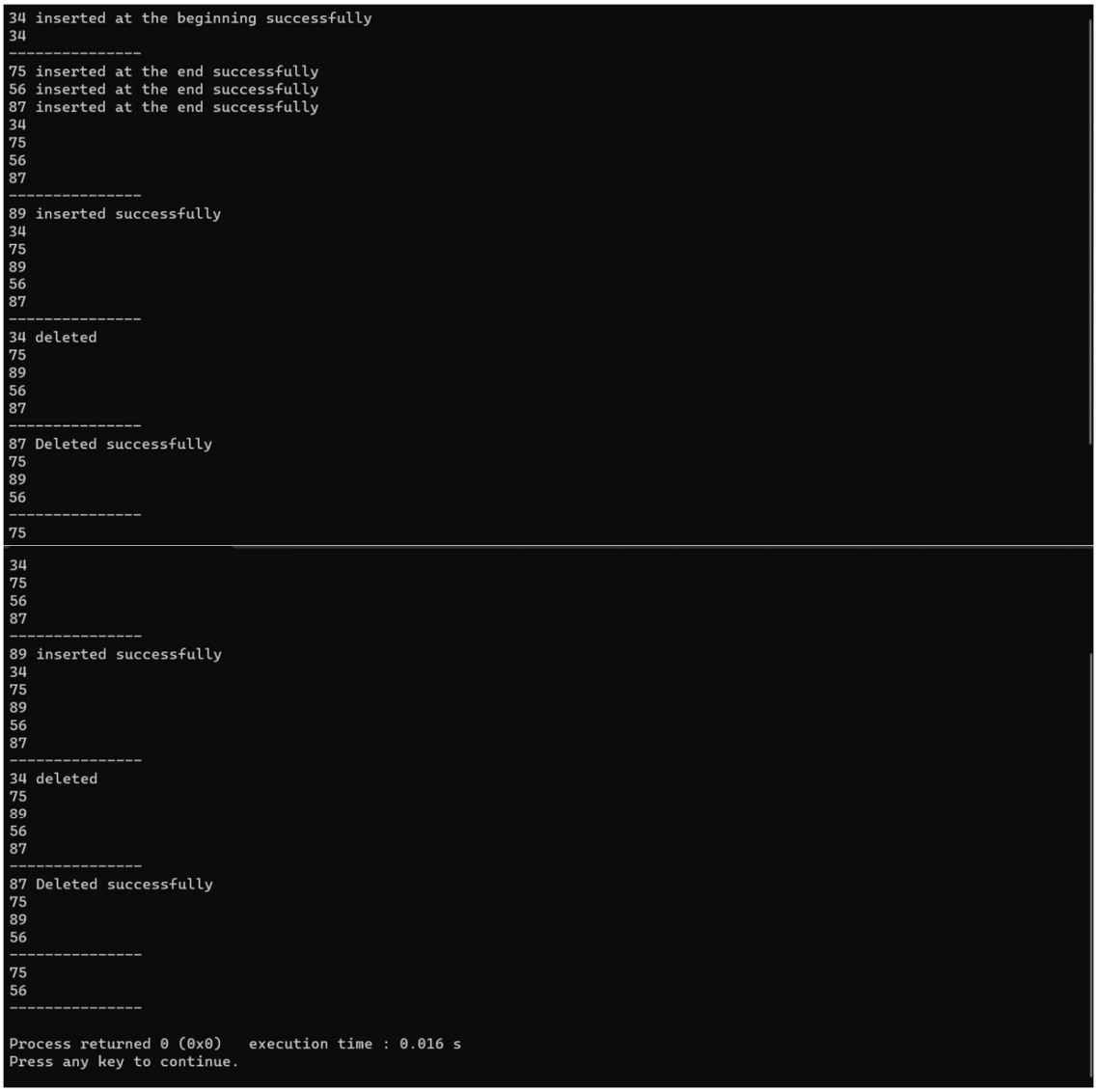
```
void display(){
if(front==-1){
    printf("Underflow");
}else{
    int i=front;
while(i!=rear){
    printf("%d", q[i]);
    i=(i+1)%N;
}}
int main(){
    int boolean=1, choice, value;
    while(boolean){
        printf("1. Enqueue\n
                2. Dequeue\n
                3. Display\n
                4. Exit");
        scanf("%d", &choice);
        switch(choice){
        case1: printf("Enter the value");
               scanf("%d", &value);
               enqueue(value);
               break;
        case2: dequeue();break;
        case3: display();break;
        case4: boolean=0;break;
        default: printf("Invalid");break;
}}}
```

Lab 3 – Linked List

```
#include <stdio.h>
#include <stdlib.h>
struct node{
    int data;
    struct node *link;
};
void countNode(struct node *head){
    int count=0;
    if(head==NULL){
        printf("Empty List");
    }else{
        struct node *ptr;
        ptr=head;
        while(ptr!=NULL){
            count++;
            ptr=ptr->link;
        }
        printf("%d", count);
    }
}
void printData(struct node* head){
    if(head==NULL){
        printf("Empty list");
    }else{
        struct node *ptr;
        ptr=head;
        while(ptr!=NULL){
            printf("%d\n",ptr->data);
            ptr=ptr->link;
        }
    }
}
void insertAtEnd(struct node *head,int value){
    struct node *temp, *ptr;
    temp=(struct node* )malloc(sizeof(struct node));
    temp->data=value;
    temp->link=NULL;
    ptr=head;
    while(ptr->link!=NULL){
        ptr=ptr->link;
    }
    ptr->link=temp;
    printf("%d inserted at the end successfully\n",value);
}
void insertBeg(struct node **head, int value){
    struct node *temp;
    temp=(struct node* )malloc(sizeof(struct node));
    temp->data=value;
    temp->link = *head;
    *head = temp;
    printf("%d inserted at the beginning successfully\n",value);
}
```

```
void addToPos(struct node *head,int value, int pos){
    struct node *ptr, *ptr2;
    ptr=head;
    ptr2=(struct node* )malloc(sizeof(struct node));
    ptr2->data=value;
    ptr2->link=NULL;
    pos--;
    while(pos!=1){
        ptr=ptr->link;
        pos--;
    }
    ptr2->link=ptr->link;
    ptr->link=ptr2;
    printf("%d inserted successfully\n",value);
}

void delBeg(struct node **head){
     if (*head == NULL) {
        printf("List is empty. Cannot delete.\n");
        return;
    }
    struct node *temp;
    temp=*head;
    *head=(*head)->link;
    printf("%d deleted\n", temp->data);
    free(temp);
}
void delEnd(struct node *head){
    if(head==NULL){
        printf("The list is empty");
    }else if(head->link==NULL){
        free(head);
        head=NULL;
    }else{
        struct node *temp, *prev;
        temp=head;
        while(temp->link!=NULL){
            prev=temp;
            temp=temp->link;
        }
        printf("%d Deleted successfully\n",temp->data);
        free(temp);
        temp=NULL;
        prev->link=NULL;


    }
}
void delAtPos(struct node **head, int pos){
    struct node *current=*head;
    struct node *previous=*head;
    if(*head==NULL){
        printf("The link is empty");
    }else if(pos==1){
```

```
        *head=current->link;
        free(current);
        current=NULL;
    }else{
        while(pos!=1){
            previous=current;
            current=current->link;
            pos--;
        }
        previous->link=current->link;
        free(current);
        current=NULL;
    }
}

int main(){
    struct node *head;
    head=NULL;
    insertBeg(&head,34);
    printData(head);
    printf("---------------\n");
    insertAtEnd(head,75);
    insertAtEnd(head,56);
    insertAtEnd(head,87);
    printData(head);
    printf("---------------\n");
    addToPos(head,89,3);
    printData(head);
    printf("---------------\n");
    delBeg(&head);
    printData(head);
    printf("---------------\n");
    delEnd(head);
    printData(head);
    printf("---------------\n");
    delAtPos(&head,2);
    printData(head);
    printf("---------------\n");
  return 0;}
```


34 inserted at the beginning successfully
34
---
75 inserted at the end successfully
56 inserted at the end successfully
87 inserted at the end successfully
34
75
56
87
---
89 inserted successfully
34
75
89
56
87
---
34 deleted
75
89
56
87
---
87 Deleted successfully
75
89
56

75
34
75
56
87
---
89 inserted successfully
34
75
89
56
87
---
34 deleted
75
89
56
87
---
87 Deleted successfully
75
89
56

75
56

Process returned 0 (0x0) execution time : 0.016 s
Press any key to continue.

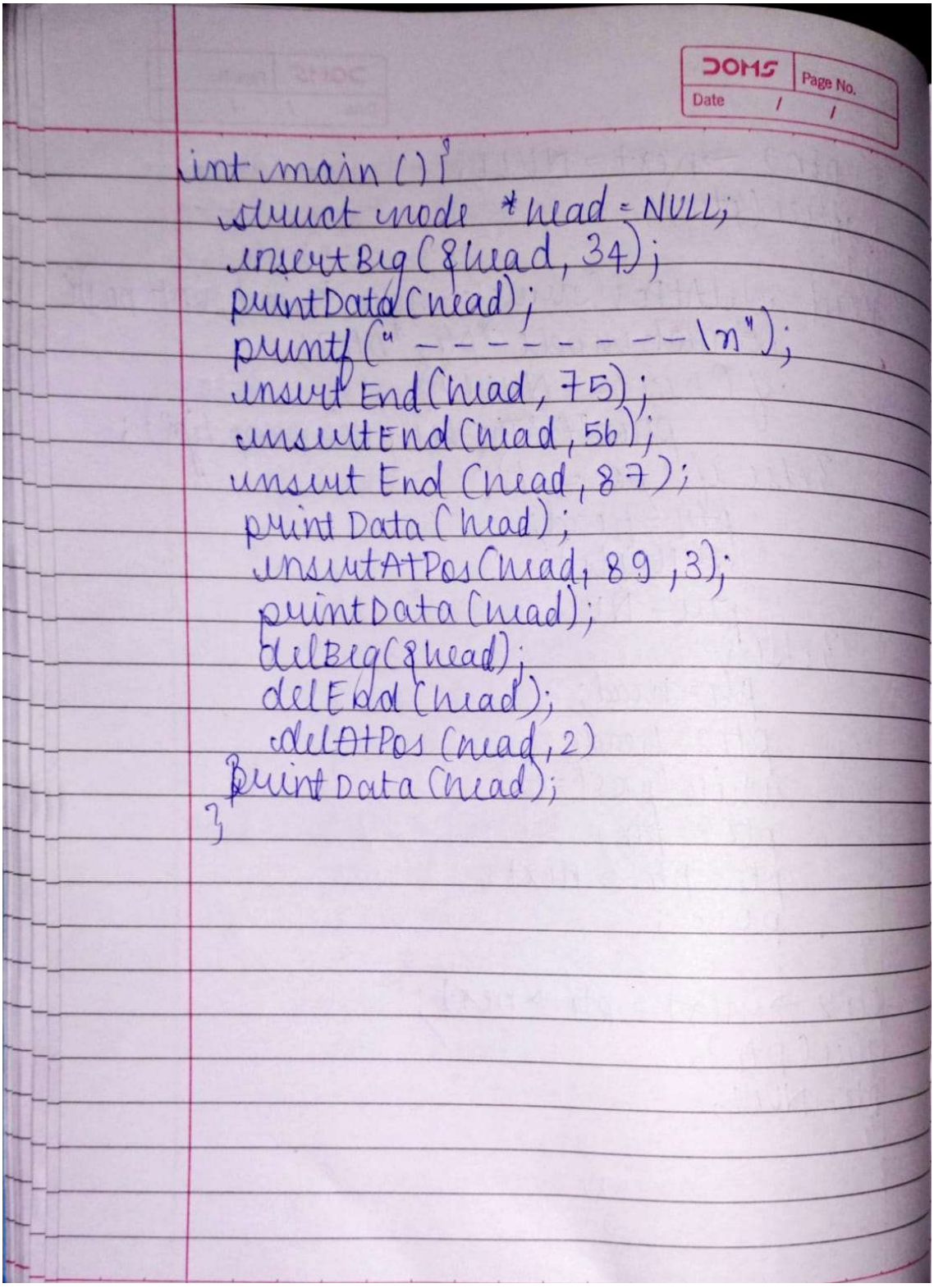
22/01/23 Linked List

```
#include <stdio.h>
#include <stdlib.h>
struct node{
    int data;
    struct node *next;
};
void printData(struct node *head){
    if (head==NULL){
        printf("The List is Empty");
    }else{
        struct node *ptr=head;
        while(ptr!=NULL){
            printf("%d \n", ptr->data);
            ptr=ptr->next; }}}
void insertBeg (struct node **head, int value)
    struct node *temp=(struct node*)
                malloc(sizeof(struct
    temp->data=value;
    temp->next=*head;
    *head=temp;
    printf("value %d added successfu
     at the beginning \n", value);
}
```

```
temp->next = ptr2->next;
ptr->next = temp;
printf("Value %d added successfully
at %d \n", value, position);}

void delBegC struct node **head){
    struct node *ptr;
    if(head==NULL){
        printf("The List is empty");
    }else{
        ptr = *head;
        *head = (*head)->next;
        free(ptr);
        ptr = NULL;
    }}
void delEnd(struct node *head){
    struct node *ptr, *ptr2;
    if(head==NULL){
        printf("The List is empty");
    }else{
        ptr = head;
        while(ptr->next != NULL){
            ptr2 = ptr;
            ptr = ptr->next;
    }
```

```
void insertEnd(struct node *head, int value){
    struct node *ptr=head;
    struct node *temp=(struct node*)
        malloc(sizeof(struct node));
    temp->data = value;
    temp->next = NULL;
    while(ptr->next!=NULL){
        ptr=ptr->next;
    }
    ptr->next=temp;
    printf("Value %d added successfully
        at the end\n", value);
}

void insertAtPos(struct node *head,
int value, int pos){
    struct node *ptr, *ptr2;
    struct node *temp=(struct node*)
        malloc(sizeof(struct
        node));
    temp->data=value;
    temp->next=NULL;
    int position=pos;
    ptr=head;
    while(pos!=1){
        ptr2=ptr;
        ptr=ptr->next;
        pos--;}
```

```
    ptr2->next = NULL;
    free(ptr);
}}
void delAtPos(struct node *head, int pos){
    struct node *ptr, *ptr2;
    if (head == NULL){
        printf("The List is empty\n");
    }else if (pos == 1){
        ptr = head;
        free(ptr);
        ptr = NULL;
    }else{
        ptr = head;
        ptr2 = head;
        while (pos != 1){
        ptr2 = ptr;
        ptr = ptr->next;
        pos--;
    }
ptr2->next = ptr->next;
free(ptr);
ptr = NULL;
}
}
```



```
int main ()
{
  struct node *head = NULL;
  insertBeg(&head, 34);
  printData(head);
  printf("----------\n");
  insertEnd(head, 75);
  insertEnd(head, 56);
  insertEnd(head, 87);
  printData(head);
  insertAtPos(head, 89, 3);
  printData(head);
  delBeg(&head);
  delEnd(head);
  delAtPos(head, 2)
  printData(head);
}
```

Output

~~Value 56 added~~

Value 34 added successfully at the beginning

34

- - - - - - -

Value 75 added successfully at the end

Value 56 added successfully at the end

Value 87 added successfully at the end

34

75

56

87

Value 89 added successfully at 3

34

75

89

56

87

- - - - - - - -

75

89

56

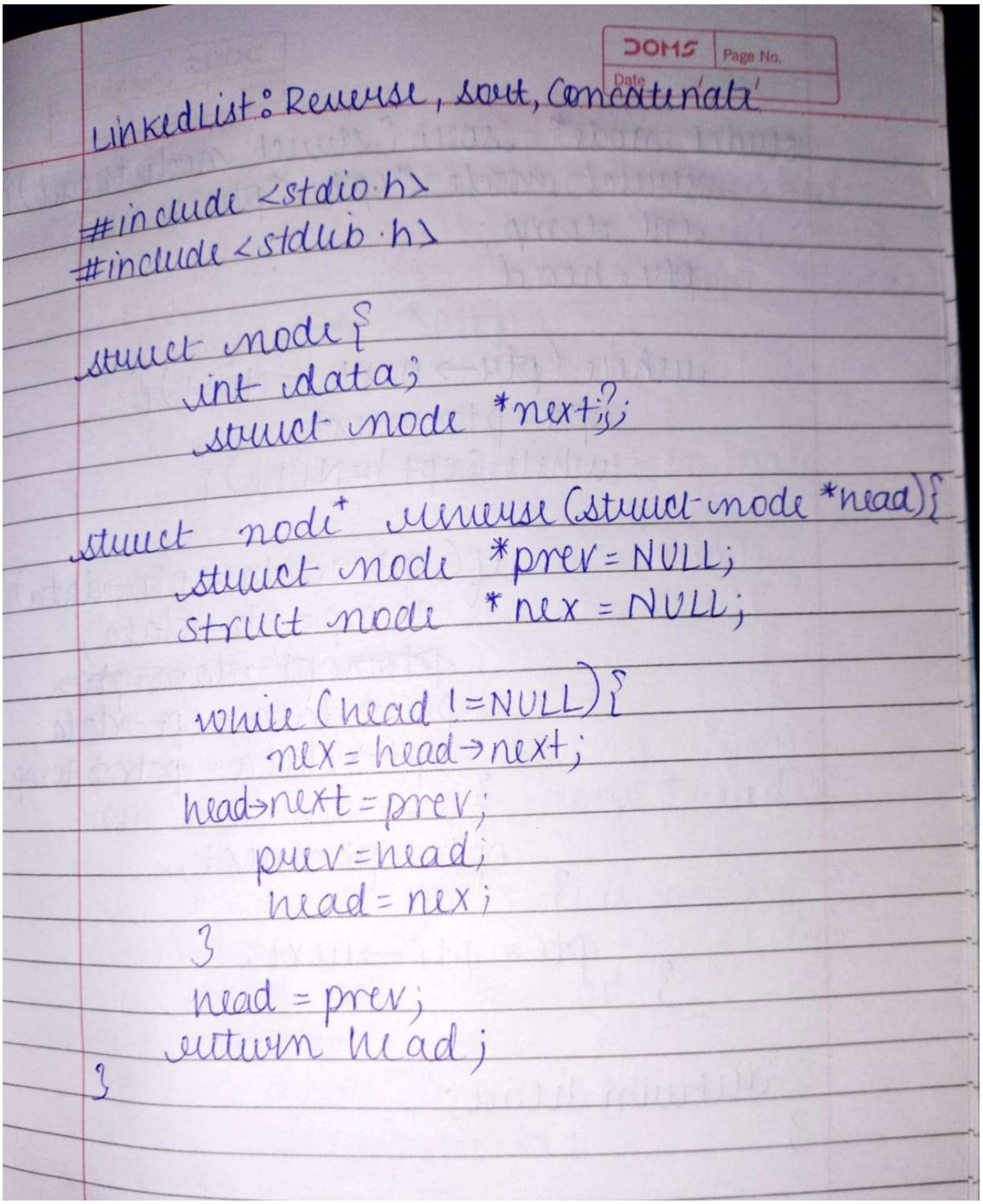
87

Lab 4 – Linked List Operations

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct node *next;
};
struct node* reverse(struct node *head){
    struct node *prev=NULL;
    struct node *nex=NULL;
    while(head!=NULL){
        nex=head->next;
        head->next=prev;
        prev=head;
        head=nex;
    }
    head=prev;
    return head;
}
void concatenate(struct node *head1, struct node *head2){
    struct node *ptr=head1;
    while(ptr->next!=NULL){
        ptr=ptr->next;
    }
    ptr->next=head2;
}
struct node* sort(struct node *head) {
    struct node *ptr, *cpt;
    int temp;
    for (ptr = head; ptr->next != NULL; ptr = ptr->next) {
        for (cpt = ptr->next; cpt != NULL; cpt = cpt->next) {
            if (cpt->data < ptr->data) {
                temp = ptr->data;
                ptr->data = cpt->data;
                cpt->data = temp;
            }
        }
    }
    return head;
}
void insertBeg(struct node **head, int value){
    struct node *temp;
    temp=(struct node *)malloc(sizeof(struct node));
    temp->data=value;
    temp->next=*head;
    *head=temp;

}
void insertEnd(struct node *head, int value){
    struct node *temp;
    struct node *ptr=head;
    temp=(struct node *)malloc(sizeof(struct node));
    temp->data=value;
```

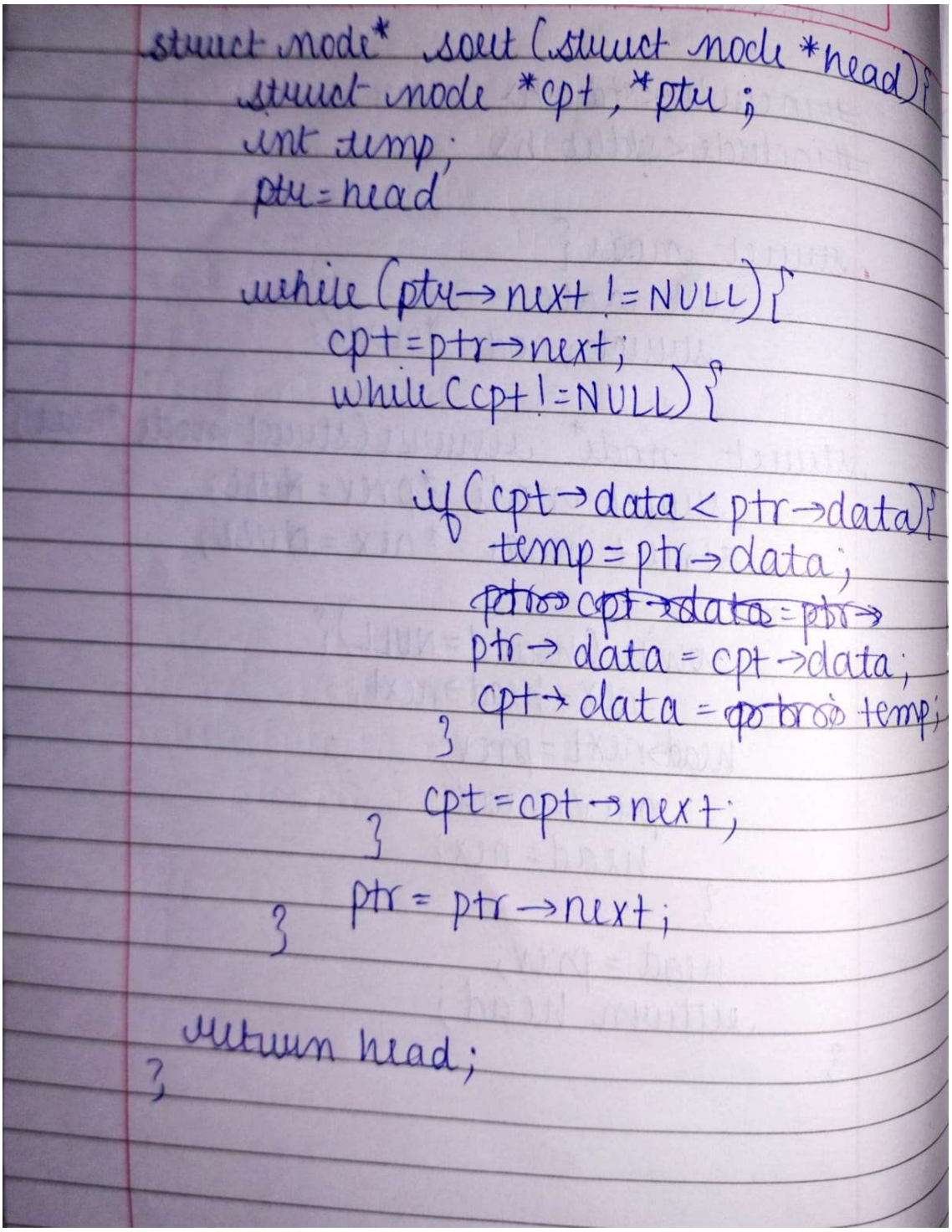
```
    temp->next=NULL;
    while(ptr->next!=NULL){
        ptr=ptr->next;
    }
    ptr->next=temp;
}
void printData(struct node *head){
    struct node *ptr;
    ptr=head;
    while(ptr!=NULL){
        printf("%d\n",ptr->data);
        ptr=ptr->next;
    }
}
int main(){
    struct node *head=NULL;
    insertBeg(&head,34);
    insertEnd(head,45);
    insertEnd(head,5);
    insertEnd(head,65);
    insertEnd(head,47);
    insertEnd(head,90);
    insertEnd(head,87);
    printData(head);
    printf("--------------------\n");
    head = reverse(head);
    printData(head);
    printf("--------------------\n");
    struct node *head1=NULL;
    struct node *head2=NULL;
    insertBeg(&head1,3);
    insertEnd(head1,4);
    insertEnd(head1,5);
    insertBeg(&head2,7);
    insertEnd(head2,6);
    insertEnd(head2,9);
    concatenate(head1,head2);
    printData(head1);
    printf("--------------------\n");
    head=sort(head);
    printData(head);
    printf("--------------------\n");

}
```

```
34
45
5
65
47
90
87
--------------------
87
90
47
65
5
45
34
--------------------
3
4
5
7
6
9
--------------------
5
34
45
47
65
87
90
--------------------
PS C:\TISSA\DSA>
```

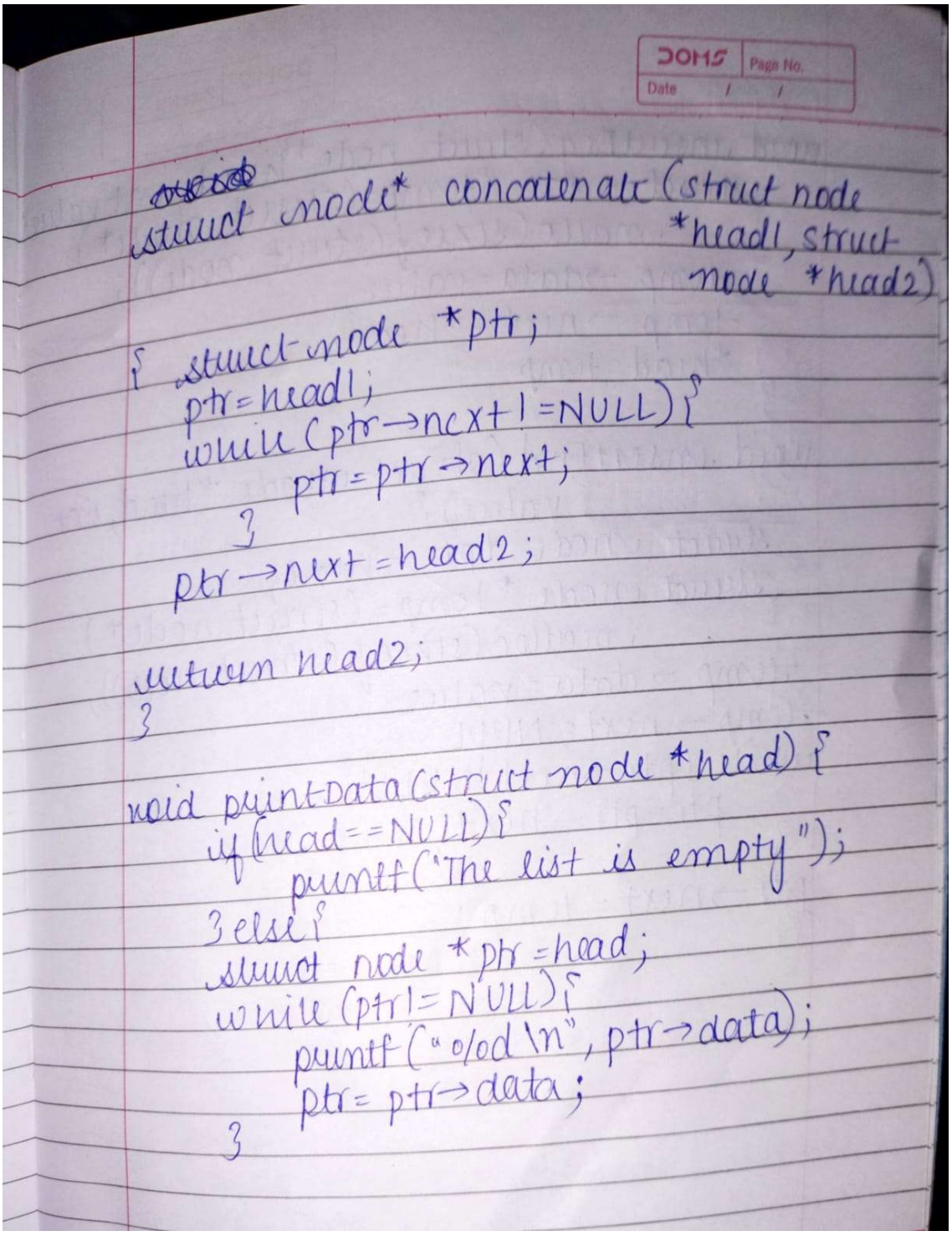


LinkedList: Reverse, Sort, Concatenate

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

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

struct node* reverse(struct node *head){
    struct node *prev = NULL;
    struct node *nex = NULL;

    while(head != NULL){
        nex = head->next;
    head->next = prev;
        prev = head;
        head = nex;
    }
    head = prev;
    return head;
}
```

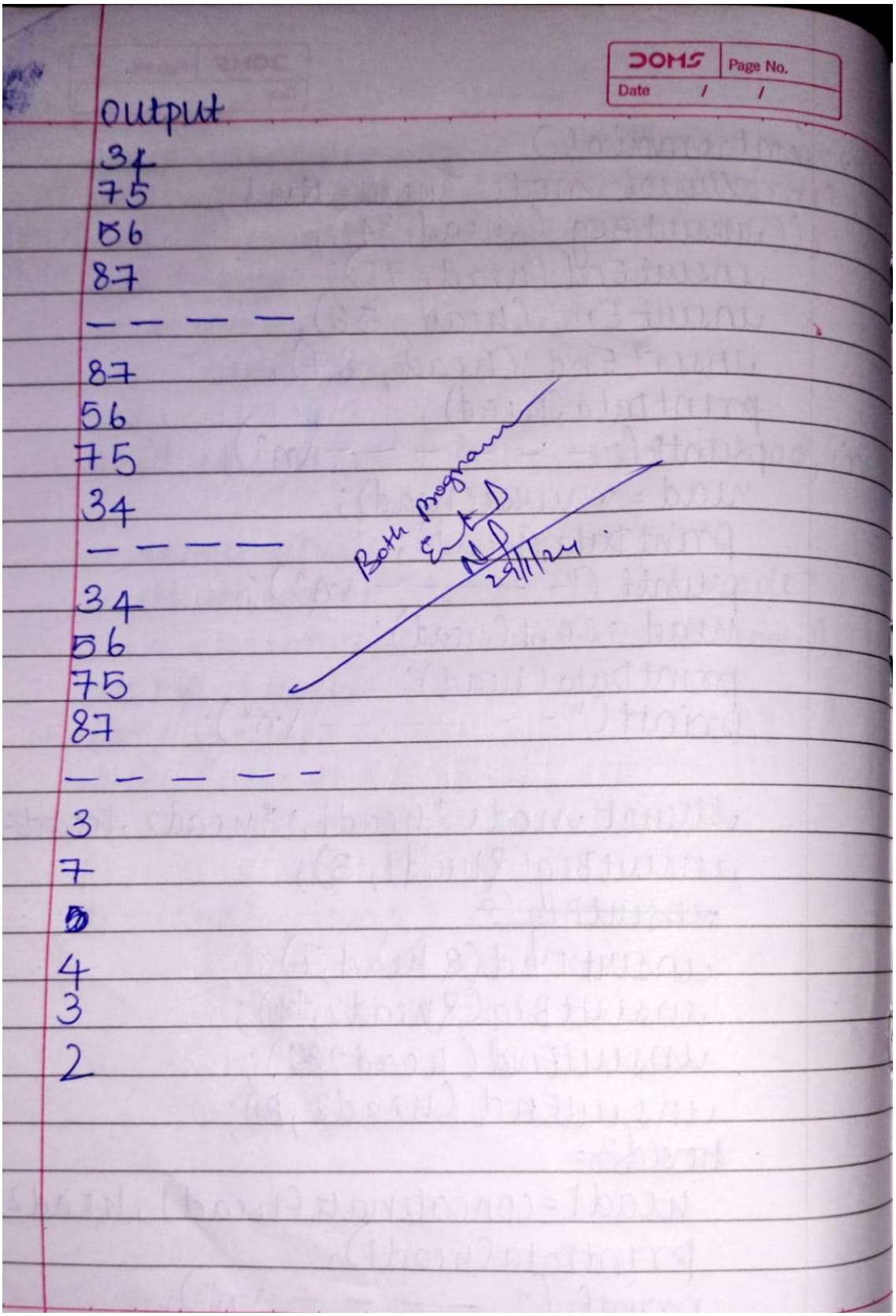


```
struct node* sort (struct node *head){
    struct node *cpt, *ptr;
    int temp;
    ptr = head

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

            if (cpt -> data < ptr -> data){
                temp = ptr -> data;
                ptr -> data = cpt -> data;
                cpt -> data = temp;
            }

            cpt = cpt -> next;
        }

        ptr = ptr -> next;
    }

    return head;
}
```



```
struct node* concatenate (struct node
                          *head1, struct
                          node *head2)
{ struct node *ptr;
  ptr = head1;
  while (ptr->next!=NULL){
      ptr = ptr->next;
  }
  ptr->next = head2;

return head2;
}

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

```
void insertBeg(struct node **head, int value)
    struct node *temp=(struct node*)
        malloc(sizeof(struct node));
    temp->data=value;
    temp->next=*head;
    *head=temp
}
void insertEnd(struct node *head, int
        value);
    struct node *ptr=head;
    struct node *temp=(struct node*)
        malloc(sizeof(struct node));
    temp->data=value
    temp->next=NULL
    while(ptr->next!=NULL){
        ptr=ptr->next;
    }
    ptr->next=temp;
}
```


DOMS
Page No.
Date / /
output
34
75
56
87
87
56
75
34
34
56
75
87
3
7
4
3
2
Both programs
Executed
29/11/24

Stacks using Linked List

```
#include <stdio.h>
#include <stdlib.h>
struct node{
    int data;
    struct node *next;
}*top=NULL;
void push(int value){
    struct node *newNode;
    newNode=(struct node *)malloc(sizeof(struct node));
    newNode->data=value;
    newNode->next=top;
    top=newNode;
    printf("Successfully Added %d",value);
}
void pop(){
    if(top==NULL){
        printf("The stack is empty");
    }else{
        struct node *temp;
    temp=top;
    top=top->next;
    printf("%d deleted\n", temp->data);
    free(temp);
    }
}
void display(){
    struct node *ptr;
    ptr=top;
    while(ptr!=NULL){
        printf("%d\n",ptr->data);
        ptr=ptr->next;
    }
}

int main(){
    int boolean=1,choice,value;
    while(boolean){
        printf("\n1.PUSH\t2.PULL\t3.DISPLAY\t4.EXIT\n");
        scanf("%d",&choice);
        switch(choice){
        case 1:{printf("Enter a value");
            scanf("%d",&value);
                push(value);
                break;}
        case 2:pop();
        break;
        case 3:display();
        break;
        case 4:boolean=0;
        break;
        default:printf("Invalid Input");
        }
```

```
    }
    return 0;
    }
```

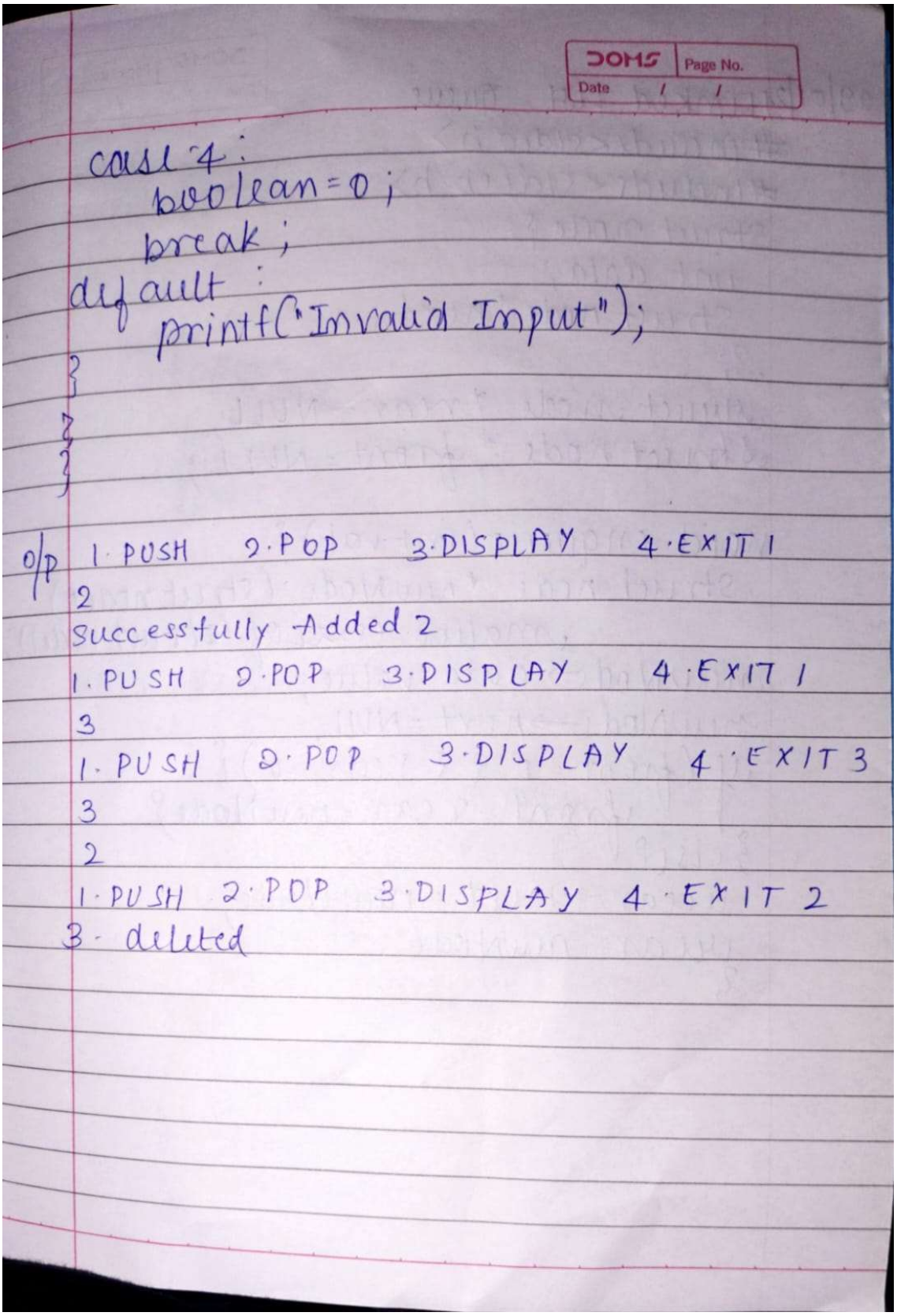
```
1.PUSH  2.PULL  3.DISPLAY       4.EXIT
1
Enter a value2
Successfully Added 2
1.PUSH  2.PULL  3.DISPLAY       4.EXIT
1
Enter a value3
Successfully Added 3
1.PUSH  2.PULL  3.DISPLAY       4.EXIT
1
Enter a value5
Successfully Added 5
1.PUSH  2.PULL  3.DISPLAY       4.EXIT
2
5 deleted

1.PUSH  2.PULL  3.DISPLAY       4.EXIT
3
3
2

1.PUSH  2.PULL  3.DISPLAY       4.EXIT
```

29/01/24

LinkedList Stacks

```
#include <stdio.h>
#include<stdlib.h>
struct node {
  int data;
  struct node *next;}
  *top = NULL;
void push(int value){
   struct node *newNode =(struct node*)
  malloc (sizeof(struct node));
  newNode->data=value;
  newNodode->next=top;
  top=newNode;
  printf("SuccessfullyAdded %d", value);
}
void pop(){
   struct node *temp;
  if(top==NULL){
     print("Underflow");
  }else{
    temp=top;
    int val=temp->data;
     top=top->next;
     free(temp);
     temp=NULL;
  printf("%d", val);}
  }
```

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

void main(){
int choice, value, boolean=1;
while(boolean){
    printf("\n1. PUSH\t2. POP\t3.
        DISPLAY\t4. EXIT");
    scanf("%d", &choice);
    switch(choice){
    case1:
        scanf("%d", &value);
        push(value);
        break;
    case2:
        pop();
        break;
        case3;
        display()
        break;
```



```
case 4:
    boolean=0;
    break;
default:
    printf("Invalid Input");
}
}
}
```

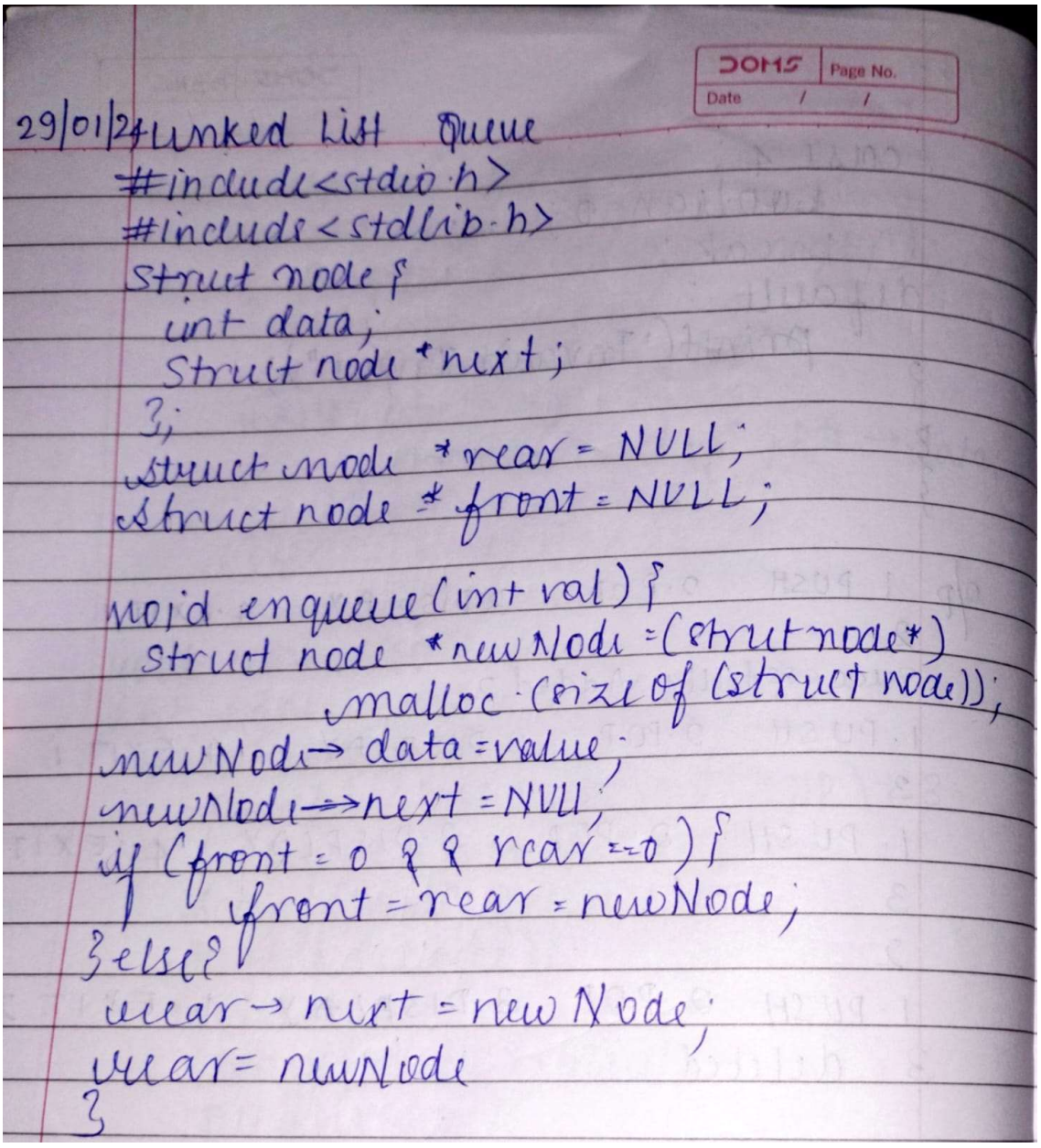
```
o/p 1.PUSH 2.POP 3.DISPLAY 4.EXIT 1
2
Successfully Added 2
1.PUSH 2.POP 3.DISPLAY 4.EXIT 1
3
1.PUSH 2.POP 3.DISPLAY 4.EXIT 3
3
2
1.PUSH 2.POP 3.DISPLAY 4.EXIT 2
3. deleted
```

Queues using Linked List

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

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

struct node *front = NULL;
struct node *rear = NULL;

void enqueue(int value) {
    struct node *newNode;
    newNode = (struct node *)malloc(sizeof(struct node));

    if (newNode == NULL) {
        printf("Memory allocation failed. Exiting...\n");
        exit(EXIT_FAILURE);
    }

    newNode->data = value;
    newNode->next = NULL;

    if (front == NULL && rear == NULL) {
        front = rear = newNode;
        printf("Successfully Added %d\n", value);
    } else {
        rear->next = newNode;
        rear = newNode;
    }
}

void dequeue() {
    if (front == NULL) {
        printf("The queue is empty\n");
    } else {
        struct node *temp;
        temp = front;
        int value = temp->data;
        front = front->next;

        printf("%d deleted\n", value);
        free(temp);
    }
}

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

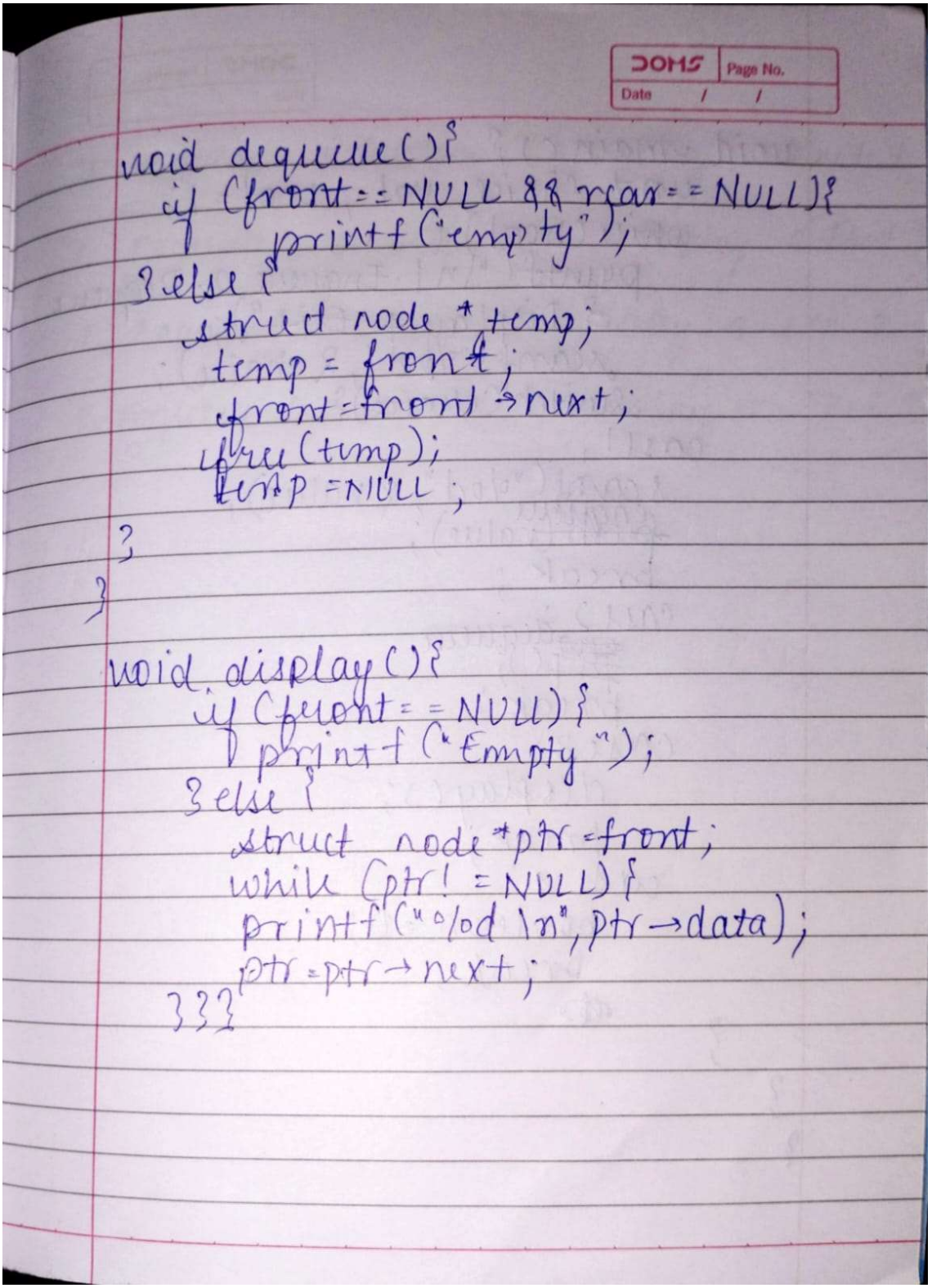
```
    }
}

int main() {
    int isRunning = 1, choice, value;

    while (isRunning) {
        printf("\n1.Enqueue\t2.Dequeue\t3.DISPLAY\t4.EXIT\n");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter a value: ");
                scanf("%d", &value);
                enqueue(value);
                break;

            case 2:
                dequeue();
                break;

            case 3:
                display();
                break;

            case 4:
                isRunning = 0;
                break;

            default:
                printf("Invalid Input\n");
        }
    }

    return 0;
}
```

```
1.Enqueue       2.Dequeue       3.DISPLAY       4.EXIT
1
Enter a value: 2
Successfully Added 2

1.Enqueue       2.Dequeue       3.DISPLAY       4.EXIT
1
Enter a value: 4

1.Enqueue       2.Dequeue       3.DISPLAY       4.EXIT
1
Enter a value: 5

1.Enqueue       2.Dequeue       3.DISPLAY       4.EXIT
2
2 deleted

1.Enqueue       2.Dequeue       3.DISPLAY       4.EXIT
3
4
5

1.Enqueue       2.Dequeue       3.DISPLAY       4.EXIT
```

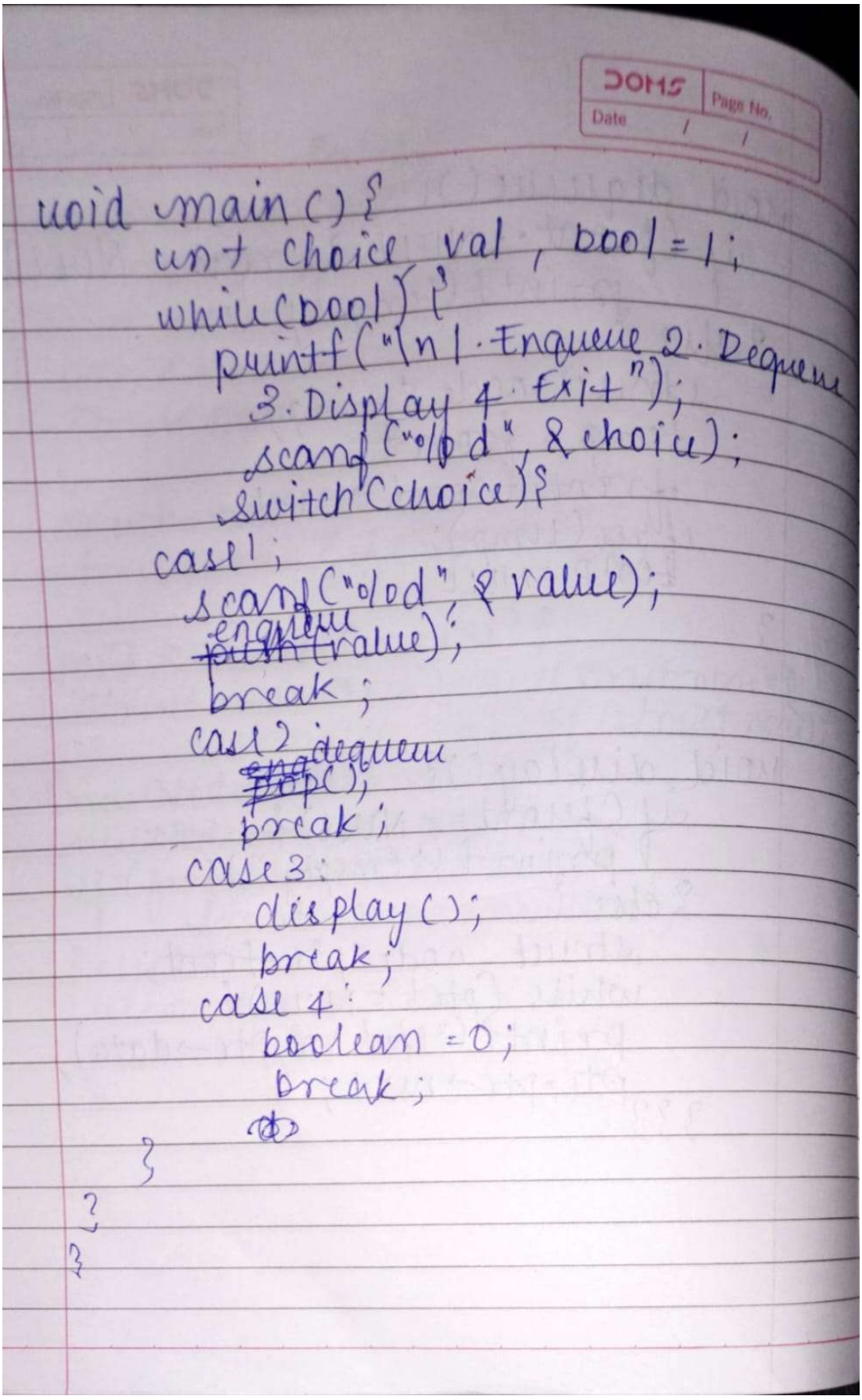


29/01/24 Linked List Queue

```
#include<stdio.h>
#include<stdlib.h>
struct node{
  int data;
  struct node *next;
};
struct node *rear = NULL;
struct node *front = NULL;

void enqueue(int val){
  struct node *newNode=(struct node*)
        malloc(sizeof(struct node));
newNode->data=value;
newNode->next=NULL;
if(front==0 && rear==0){
    front=rear=newNode;
}else{
 rear->next=newNode;
 rear=newNode
}
```



```
void dequeue(){
  if (front == NULL && rear == NULL){
     printf("empty");
}else{
   struct node *temp;
    temp = front;
    front = front->next;
   free(temp);
    temp = NULL;
}
}

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



```
void main(){
    int choice val, bool=1;
    while(bool){
      printf("\n1. Enqueue 2. Dequeue
        3. Display 4. Exit");
      scanf("%d", &choice);
      switch(choice){
case 1;
  scanf("%d", &value);
  enqueue(value);
  break;
case 2 dequeue
  pop();
  break;
case 3:
  display();
  break;
case 4:
  boolean = 0;
  break;
    }
}
```

Lab 5 Double Linked Lists

```
#include <stdio.h>
#include <stdlib.h>
struct node{
    struct node *prev;
    int data;
    struct node *next;
};
struct node* DInsertAtBeg(struct node *head, int value){
    struct node *temp;
    temp=(struct node *)malloc(sizeof(struct node));
    temp->prev=NULL;
    temp->next=NULL;
    temp->data=value;
    if(head==NULL){
        head=temp;

    }else{
        temp->next=head;
        head->prev=temp;
        head=temp;

    }
    printf("Successfully Added %d at the beginning\n",value);
    return head;
}
struct node* DInsertAtEnd(struct node *head, int value){
    struct node *temp;
    temp=(struct node *)malloc(sizeof(struct node));
    temp->prev=NULL;
    temp->next=NULL;
    temp->data=value;
    if(head==NULL){
        head=temp;

    }else{
        struct node *ptr;
        ptr=head;
        while(ptr->next!=NULL){
            ptr=ptr->next;
        }
        ptr->next=temp;
        temp->prev=ptr;


    }
    printf("Successfully Added %d at the end\n",value);
    return head;
}
struct node* DInsertAfter(struct node* head, int pos, int value){
    struct node *temp;
    temp=(struct node *)malloc(sizeof(struct node));
    temp->prev=NULL;
```

```
    temp->next=NULL;
    temp->data=value;
    int position = pos;
    struct node *ptr1, *ptr2;
    pos--;
    ptr1=head;

    while(pos!=0){
        ptr1=ptr1->next;
        pos--;
    }
    ptr2=ptr1->next;
    temp->prev=ptr1;
    temp->next=ptr2;
    ptr1->next=temp;
    ptr2->prev=temp;
    printf("Successfully Added %d after position %d\n",value,position);
    return head;

}
struct node* DInsertBefore(struct node* head, int pos, int value){
    struct node *temp;
    temp=(struct node *)malloc(sizeof(struct node));
    temp->prev=NULL;
    temp->next=NULL;
    temp->data=value;
    int position=pos;
    struct node *ptr1, *ptr2;
    pos--;
    ptr1=head;

    while(pos!=1){
        ptr1=ptr1->next;
        pos--;
    }
    ptr2=ptr1->next;
    temp->prev=ptr1;
    temp->next=ptr2;
    ptr1->next=temp;
    ptr2->prev=temp;
    printf("Successfully Added %d before position %d\n",value,position);
    return head;

}
void delAtBeg(struct node **head){
    if(*head==NULL){
        printf("The List is Empty.\n");
    }else{
        struct node *ptr=*head;
        *head=(*head)->next;
       (*head)->prev = NULL;
        free(ptr);
        ptr=NULL;
        printf("Value deleted at the beginning\n");
```

```
    }
}
void delAtEnd(struct node *head){
    if(head==NULL){
        printf("The List is Empty.\n");
    }else{
        struct node *ptr=head;
        while(ptr->next!=NULL){
            ptr=ptr->next;
        }
        ptr->prev->next=NULL;
        free(ptr);
        ptr=NULL;
        printf("Value deleted at the end\n");
    }
}
void delVal(struct node *head, int val){
    if(head==NULL){
        printf("Empty list");
    }else{
        struct node *ptr1,*ptr2;
        ptr1=head;
        while(ptr1->data!=val){
            ptr1=ptr1->next;
        }
        if(ptr1!=NULL){
        ptr2=ptr1->next;
        ptr2->prev=ptr1->prev;
        ptr1->prev->next=ptr2;
        free(ptr1);
        ptr1=NULL;
        printf("%d deleted\n",val);}
        else{
            printf("Not Found\n");
        }
    }
}
void printData(struct node* head){
    if(head==NULL){
        printf("Empty list\n");
    }else{
        struct node *ptr;
        ptr=head;
        while(ptr!=NULL){
            printf("%d\n",ptr->data);
            ptr=ptr->next;
        }
        printf("----------------\n");
    }
}
int main(){
    struct node *head;
    head=NULL;
    printData(head);
```

```
    head=DInsertAtBeg(head,30);
    head=DInsertAtBeg(head,45);
    head=DInsertAtEnd(head,56);
    printData(head);
    head=DInsertAfter(head,2,23);
    printData(head);
    head=DInsertBefore(head,2,11);
    printData(head);
    delAtBeg(&head);
    printData(head);
    delAtEnd(head);
    printData(head);
    delVal(head,30);
    printData(head);
    return 0;
}
```

```
Successfully Added 30 at the beginning
Successfully Added 45 at the beginning
Successfully Added 56 at the end
45
30
56
----------------
Successfully Added 23 after position 2
45
30
23
56
----------------
Successfully Added 11 before position 2
45
11
30
23
56
----------------
Value deleted at the beginning
11
30
23
56
----------------
Value deleted at the end
11
30
23
----------------
30 deleted
11
23
```

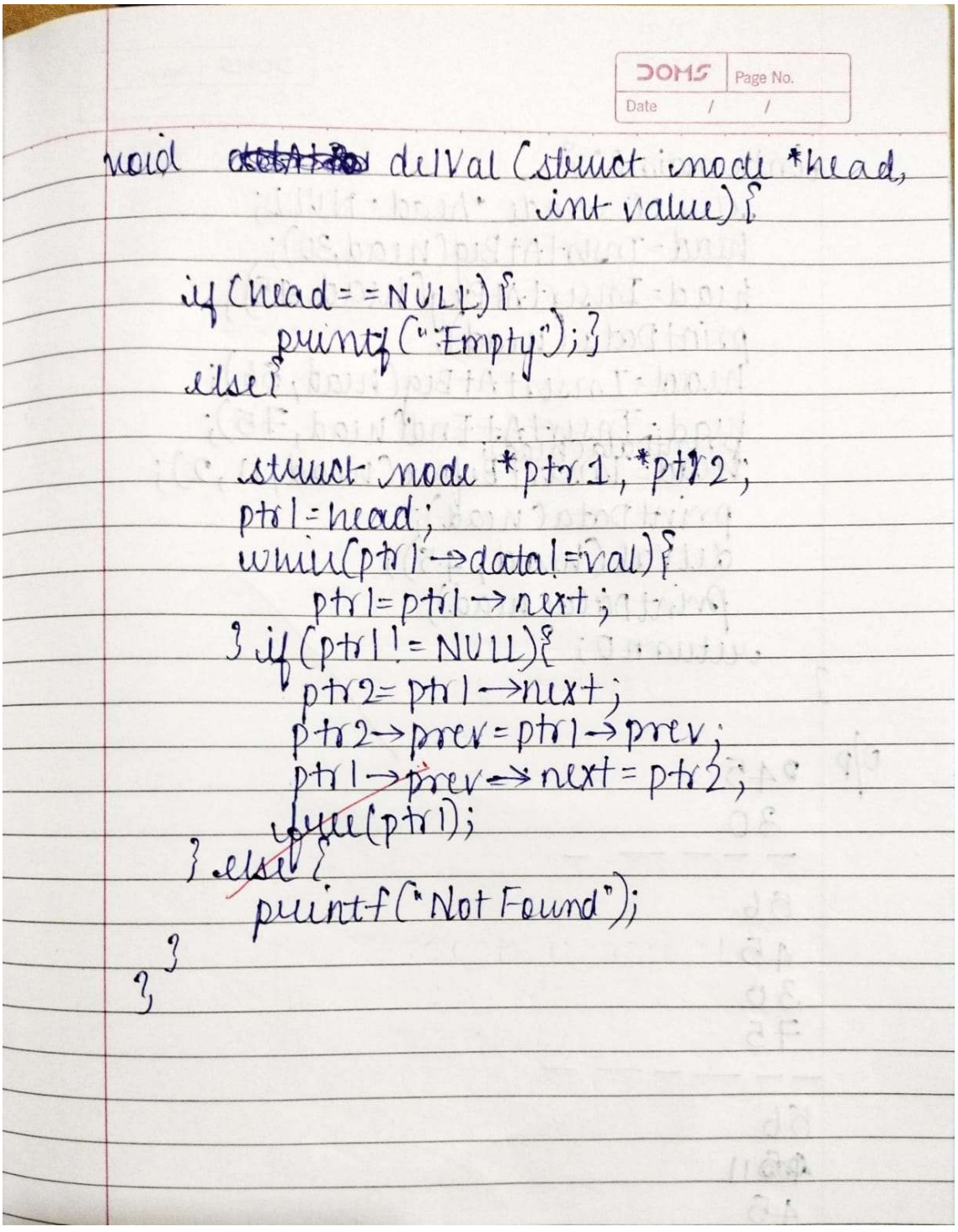
05/02/24 creation of doubly linked list, deletion buf
and display etc

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    struct node *prev;
    struct node *next;
    int data;
};
struct node* insertAtBeg(struct node
                         *head, int val
    struct node *temp;
    temp = (struct node*) malloc
                (sizeof(struct node));
    temp->next = NULL;
    temp->prev = NULL;
    temp->data = val;

    if (head == NULL) {
        head = temp;
        return head;
    } else {
        temp->next = head;
        head->prev = temp;
        head = temp;
        return head;
    }
}
```

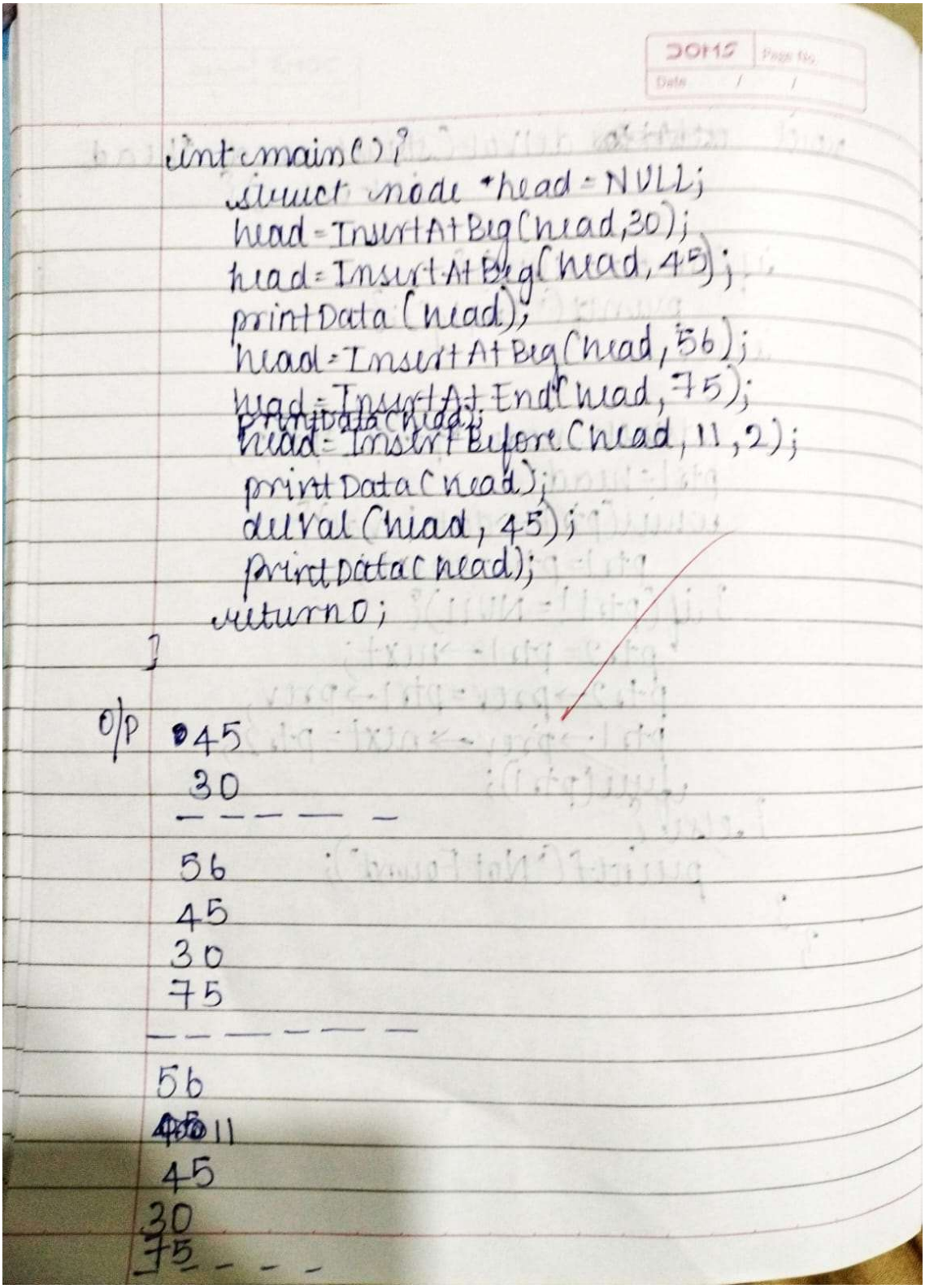
05/02/24

```
struct node * insertEnd(struct node *head,
int value){
    struct node *temp;
    temp = (struct node*)malloc(sizeof(struct node));
    temp->prev = null;
    temp->data = value;
    temp->next = null;

    if (head == NULL){
        head = temp;}
    else { struct node *ptr = head;
        temp->next = head;
        while (ptr->next != NULL){
            head->prev = temp; ptr = ptr->next;
            head = temp ; } temp->prev = ptr
                          ptr->next = temp
    }
    return head;
}
void printData(struct node *head){
    struct node *ptr = head;
    while(ptr != NULL){
        printf("%d\n" ptr->data);
        ptr = ptr->next;
    }
    printf(" ------ ");
}
```

```
struct node *insertBefore(struct node
*head, int value, int pos){

    struct node *temp;
    temp = (struct node *)malloc(sizeof
                                 (struct node));

  temp->prev = NULL;
  temp->data = value;
  temp->next = NULL;
  struct node *ptr1, *ptr2;
   pos--;
   while(pos!=1){
       ptr1 = ptr1->next;
       pos--;
   }
  ptr2 = ptr1->next;
  temp->prev = ptr1;
      temp->next = ptr2;
     ptr1->next = temp;
     ptr2->prev = temp;
    return head;
}
```



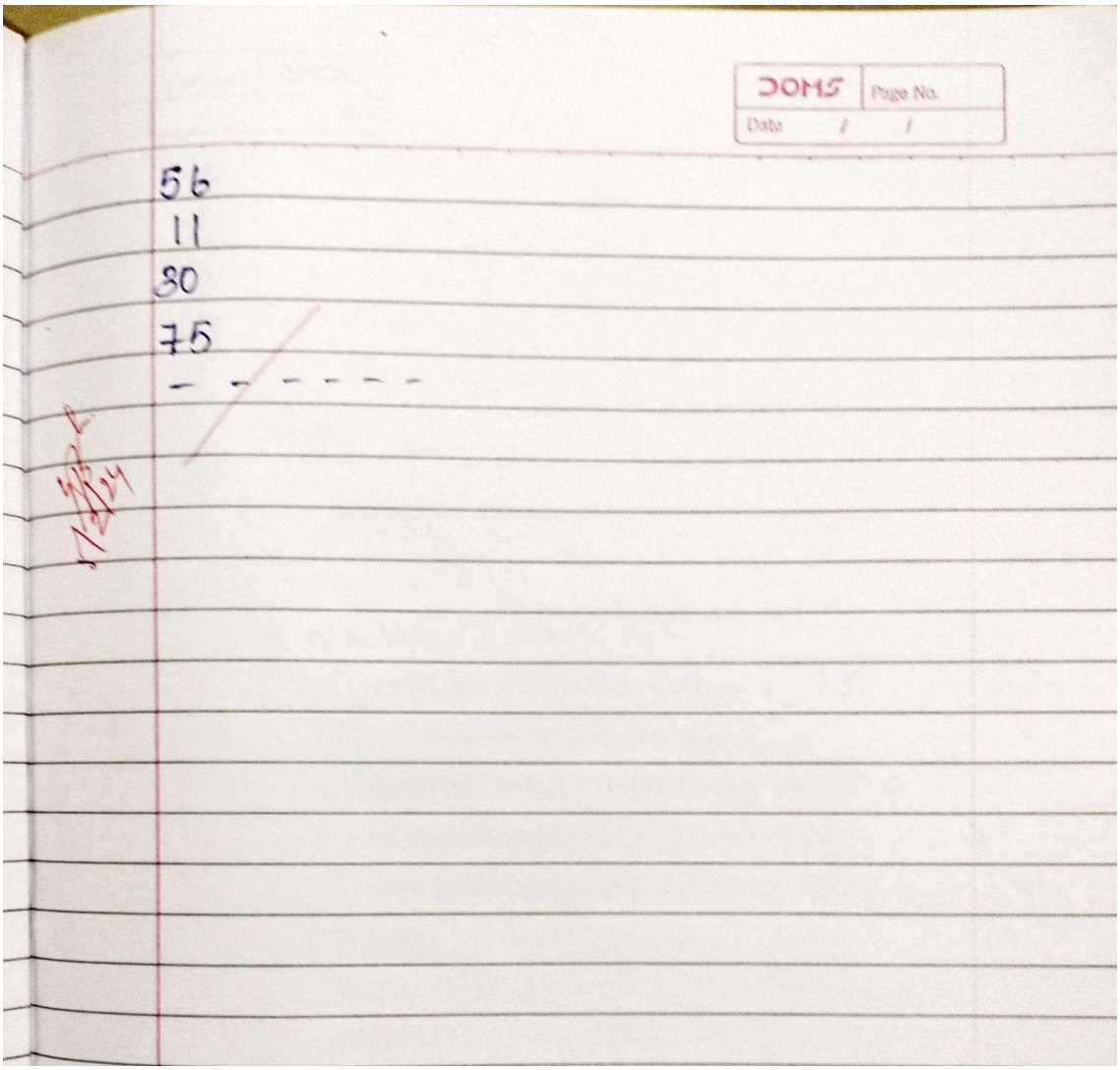
```
void delVal(struct node *head,
                int value){

  if(head==NULL){
     printf("Empty");}
  else{

     struct node *ptr1,*ptr2;
     ptr1=head;
     while(ptr1->data!=val){
        ptr1=ptr1->next;
     } if(ptr1!=NULL){
        ptr2=ptr1->next;
        ptr2->prev=ptr1->prev;
        ptr1->prev->next=ptr2;
        free(ptr1);
  } else{
     printf("Not Found");
}}
```



```
int main()
{
    struct node *head = NULL;
    head = InsertAtBig(head,30);
    head = InsertAtBig(head, 45);
    printData(head);
    head = InsertAtBig(head, 56);
    head = InsertAtEnd(head, 75);
    head = InsertBefore(head, 11, 2);
    printData(head);
    delVal(head, 45);
    printData(head);
    return 0;
}
```

O/P

```
45
30
-----
56
45
30
75
-------
56
11
45
30
75
----
```


DOMS
Page No.
Date / /
56
11
80
75

Lab 6 – Binary Search Tree

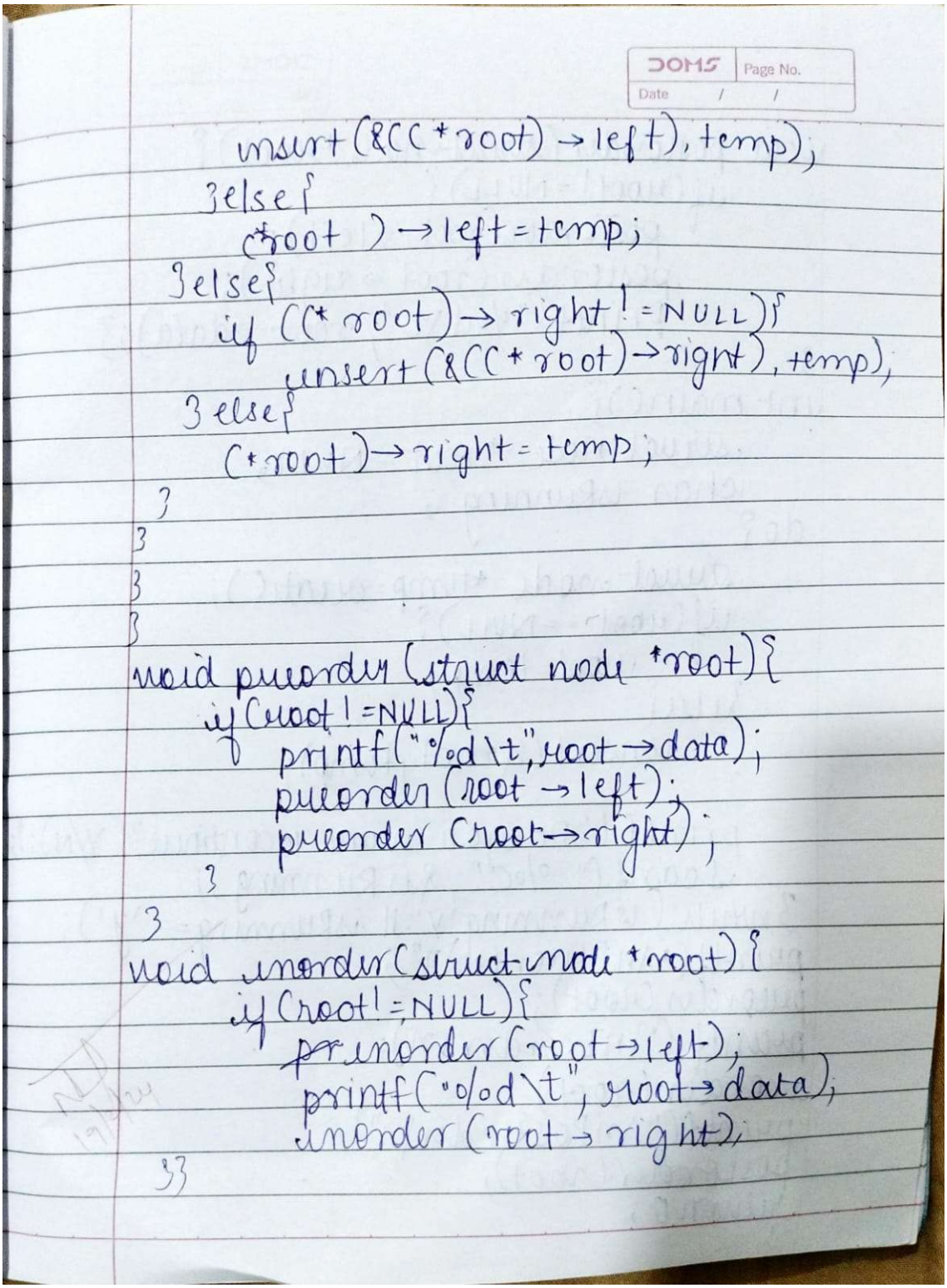
```
#include<stdio.h>
#include<stdlib.h>

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

struct node* create() {
    struct node *temp;
    temp = (struct node *)malloc(sizeof(struct node));
    printf("Enter value: ");
    int value;
    scanf("%d", &value);
    temp->data = value;
    temp->left = temp->right = NULL;
    return temp;
}

void insert(struct node **root, struct node *temp) {
    if (*root == NULL) {
        *root = temp;
    } else {
        if ((*root)->data > temp->data) {
            if ((*root)->left != NULL) {
                insert(&((*root)->left), temp);
            } else {
                (*root)->left = temp;
            }
        } else {
            if ((*root)->right != NULL) {
                insert(&((*root)->right), temp);
            } else {
                (*root)->right = temp;
            }
        }
    }
}

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

void inorder(struct node *root) {
    if (root != NULL) {
        inorder(root->left);
```

```
        printf("%d\t", root->data);
        inorder(root->right);
    }
}

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

int main() {
    struct node *root = NULL;
    char isRunning;
    do {
        struct node *temp = create();
        if (root == NULL) {
            root = temp;
        } else {
            insert(&root, temp);
        }
        printf("Do you want to continue? (Y/N): ");
        scanf(" %c", &isRunning);
    } while (isRunning == 'Y');

    printf("\nPreorder\n");
    preorder(root);
    printf("\nInorder\n");
    inorder(root);
    printf("\nPostorder\n");
    postorder(root);

    return 0;
}
```

```
PS C:\Users\Tissa Maria> cd "c:\TISSA\Data Structures 2023-24\BST, LeetCode\" ; if ($?) { gcc BST.c -o BST } ; if ($?) { .\BST }
Enter value: 45
Do you want to continue? (Y/N): Y
Enter value: 87
Do you want to continue? (Y/N): Y
Enter value: 23
Do you want to continue? (Y/N): Y
Enter value: 98
Do you want to continue? (Y/N): Y
Enter value: 43
Do you want to continue? (Y/N): Y
Enter value: 20
Do you want to continue? (Y/N): Y
Enter value: 12
Do you want to continue? (Y/N): N

Preorder
45      23      20      12      43      87      98
Inorder
12      20      23      43      45      87      98
Postorder
12      20      43      23      98      87      45
PS C:\TISSA\Data Structures 2023-24\BST, LeetCode>
```

19/02/24 Binary Search Tree
Creation, Traversal

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



```
        insert(&((*root)->left), temp);
    }else{
        (*root)->left = temp;
    }else{
    if ((*root)->right != NULL){
        insert(&((*root)->right), temp);
    }else{
        (*root)->right = temp;
    }
}
}
}
void preorder(struct node *root){
    if(root != NULL){
        printf("%d\t", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}
void inorder(struct node *root){
    if(root != NULL){
        preinorder(root->left);
        printf("%d\t", root->data);
        inorder(root->right);
}}
```

```
void postorder(struct node *root){
    if(root!=NULL){
        postorder(root->left);
        postorder(root->right);
        printf("%d\t",root->data);}
}
int main(){
    struct node *root=NULL;
    char isRunning;
do{
    struct node *temp=create();
    if(root==NULL){
        root=temp;
    }else{
        insert(&root,temp);
    }
    printf("Do you want to continue?(Y/N)
    scanf("%c",&isRunning);
}while(isRunning=='y'||isRunning=='y');
printf("\nPreorder\n");
preorder(root);
printf("\nInorder\n");
inorder(root);
printf("\nPostorder\n");
postorder(root);
return 0;
}
```

Lab 7 – Graphs

BFS

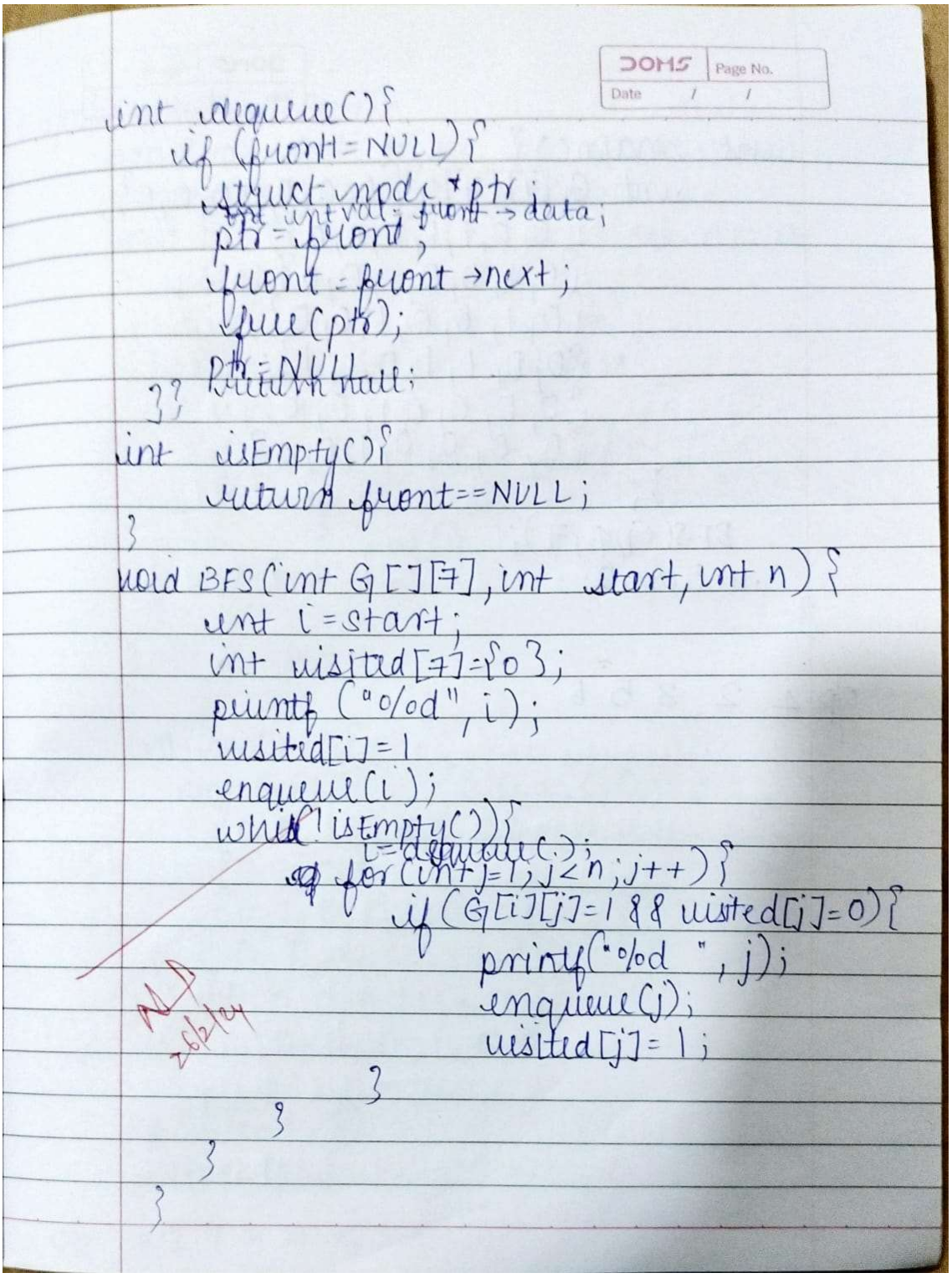
```
#include <stdio.h>
#include<stdlib.h>
//for queue
struct node{
    int data;
    struct node *next;
};
struct node *front=NULL;
struct node *rear=NULL;

void enqueue(int x){
    struct node *newNode;
    newNode=(struct node *)malloc(sizeof(struct node));
    newNode->data=x;
    newNode->next=NULL;
    if(front==NULL&&rear==NULL){
        front=rear=newNode;
    }else{
        rear->next=newNode;
        rear=newNode;
    }
}
int dequeue(){
    if(front!=NULL){
    struct node *ptr;
    ptr=front;
    front=front->next;
    free(ptr);
    ptr=NULL;}
}
int isEmpty(){
    return front==NULL;
}
void BFS(int G[][7],int start,int n){
    int i=start;

    int visited[7]={0};
    printf("%d ",i);
    visited[i]=1;
    enqueue(i);
    while(!isEmpty()){
        i=dequeue();
        for(int j=1;j<n;j++){
            if(G[i][j]=1&&visited[j]==0){
                enqueue(j);
                visited[j]=1;
                printf("%d ",j);
            }
        }
    }
```

```
}
int main(){
    int G[7][7]={{0,0,0,0,0,0,0},
    {0,0,1,1,0,0,0},
    {0,1,0,0,1,0,0},
    {0,1,0,0,1,0,0},
    {0,0,1,1,0,1,1},
    {0,0,0,0,1,0,0},
    {0,0,0,0,1,0,0}

    };
    printf("BFS\n");
    BFS(G,4,7);


    return 0;
}
```

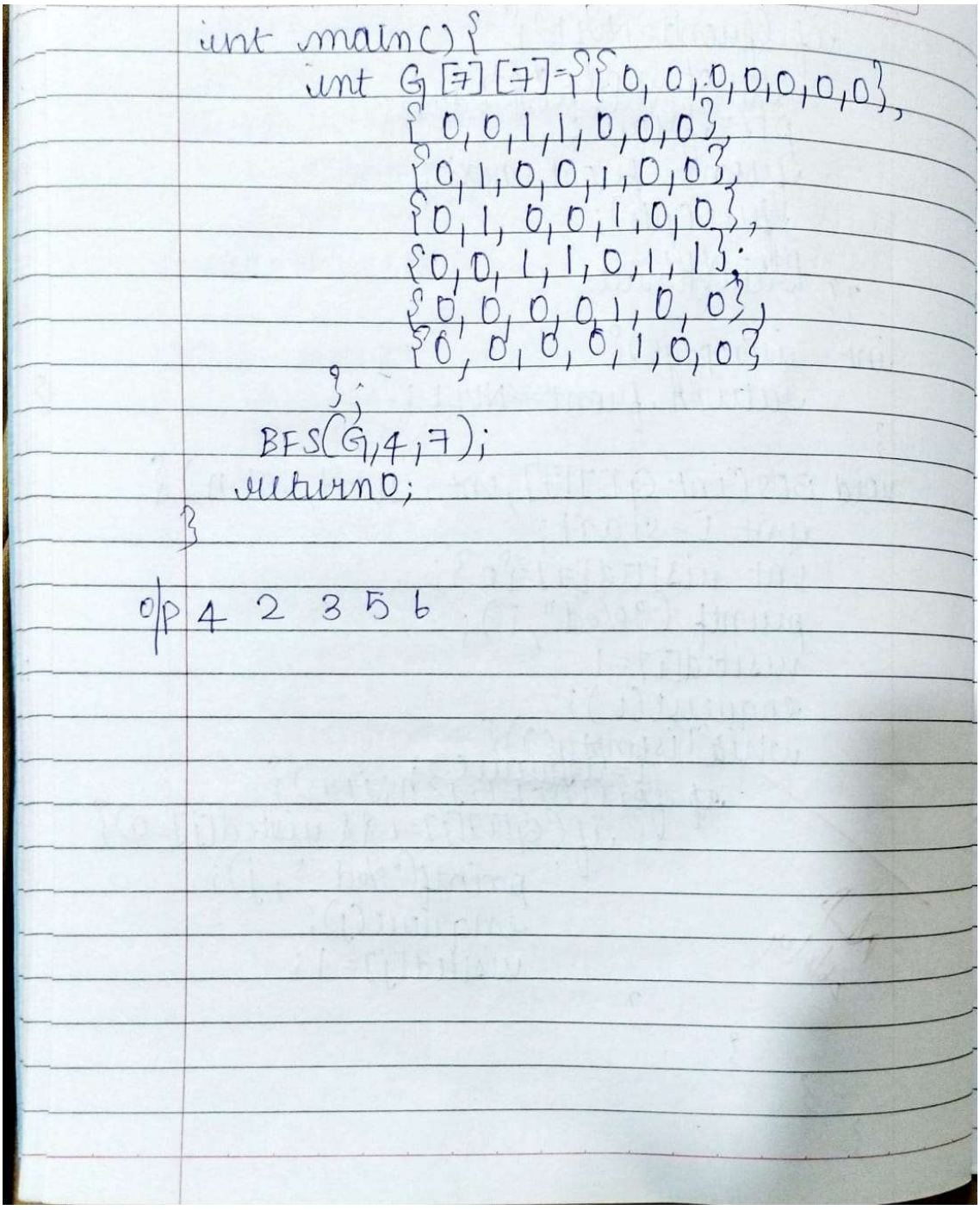
```
PS C:\TISSA\Data Structures 2023-24\Graphs, LeetCode> cd "c:\TISSA\Data Structures 2023-24\Graphs, LeetCode\" ; if ($?) {
gcc BreadthFirstSearch.c -o BreadthFirstSearch } ; if ($?) { .\BreadthFirstSearch }
BFS
4 1 2 3 5 6
```

26/02/24 Breadth First Search

```
#include <stdio.h>
#include <stdlib.h>
struct node{
    int data;
    struct node *next;
}*rear=NULL, *front=NULL;

void enqueue(int x){
    struct *newNode;
    newNode (struct node *)malloc(sizeof(
                                    struct Node));
    newNode->data = x;
    newNode->next = NULL;
    if(newNode
    if(front == NULL && rear == NULL){
            front = rear = newNode;
    }else{
        rear->next = newNode;
        rear = newNode;
    }}
```



```
int dequeue(){
    if (front != NULL){
        struct node *ptr;
        int interval; front -> data;
        ptr = front;
        front = front -> next;
        free(ptr);
        ptr = NULL;
    }
}
int isEmpty(){
    return front == NULL;
}
void BFS(int G[][7], int start, int n){
    int i = start;
    int visited[7] = {0};
    printf("%d", i);
    visited[i] = 1
    enqueue(i);
    while(!isEmpty()){
        i = dequeue();
        for(int j = 1; j < n; j++){
            if(G[i][j] = 1 && visited[j] = 0){
                printf("%d ", j);
                enqueue(j);
                visited[j] = 1;
            }
        }
    }
}
```



```
int main(){
    int G[7][7]={{0,0,0,0,0,0,0},
                 {0,0,1,1,0,0,0},
                 {0,1,0,0,1,0,0},
                 {0,1,0,0,1,0,0},
                 {0,0,1,1,0,1,1},
                 {0,0,0,0,1,0,0},
                 {0,0,0,0,1,0,0}
                };
    BFS(G,4,7);
    return 0;
}
```

o/p 4 2 3 5 6

Depth First Search

```
#include <stdio.h>
#include<stdlib.h>
void DFS(int G[][7],int start, int n){
    static int visited[7]={0};
    if(visited[start]==0)
    {
        printf("%d ", start);
        visited[start]=1;
        for(int j=1;j<n;j++){
            if(G[start][j]=1&&visited[j]==0){
                DFS(G,j,n);
            }
        }
    }
}
int main(){
    int G[7][7]={{0,0,0,0,0,0,0},
    {0,0,1,1,0,0,0},
    {0,1,0,0,1,0,0},
    {0,1,0,0,1,0,0},
    {0,0,1,1,0,1,1},
    {0,0,0,0,1,0,0},
    {0,0,0,0,1,0,0}

    };
    DFS(G,4,7);



    return 0;
}
```

```
PS C:\TISSA\Data Structures 2023-24\Graphs, LeetCode> cd "c:\TISSA\Data Structures 2023-24\Graphs, LeetCode\" ; if ($?) {
gcc DepthFirstSearch.c -o DepthFirstSearch } ; if ($?) { .\DepthFirstSearch }
4 1 2 3 5 6
PS C:\TISSA\Data Structures 2023-24\Graphs, LeetCode>
```

26/02/22 Depth First Search

```
#include <stdio.h>
#include <stdio.h>
void DFS(int G[][7], int start, int n){
    static int visited[7]={0};
    if (visited[start]==0)
    { printf("%d ", start);
    visited[start]=1);
    for (int j=1;j<n;j++){
        if(G[start][j]!=1&& visited[j]==0){
            DFS(G,j,n);
        }
    }
}
}
int main(){
    int G[7][7]={{0,0,0,0,0,0,0},
        {0,0,1,1,0,0,0},
        {0,1,0,0,1,0,0},
        {0,1,0,0,1,0,0},
        {0,0,1,1,0,1,1},
        {0,0,0,0,1,0,0},
        {0,0,0,0,1,0,0}
    };
DFS(G,4,7);
return 0;
}
```

o/p 4 2 1 3 5 6

Lab 8 – Hash Table

```
#include <stdio.h>
#define max 10
int hashTable[max] = {0};
void insertIntoHashTable(int key, int m) {
    int hashValue = key % m;
    while (hashTable[hashValue] != 0) {
        hashValue = (hashValue + 1) % m;
    }
    hashTable[hashValue] = key;
}
void displayHashTable(int m) {
    printf("\nHash Table:\n");
    printf("Address\tKey\n");
    for (int i = 0; i < m; i++) {
        printf("%d\t", i);
        if (hashTable[i] != 0) {
            printf("%d", hashTable[i]);
        }
        printf("\n");
    }
}

int main() {
    int m = max;
    int employeeKeys[] = {1234, 5678, 9876, 4321, 8765};
    for (int i = 0; i < sizeof(employeeKeys) / sizeof(employeeKeys[0]);
i++) {
        insertIntoHashTable(employeeKeys[i], m);
    }
    displayHashTable(m);
    return 0;
}
```

```
PS C:\TISSA\Data Structures 2023-24\Hash Table> cd "c:\TISSA\Data Structures 2023-24\Hash Table\" ; if ($?) { gcc hashtable.c -o hashtable
} ; if ($?) { .\hashtable }

Hash Table:
Address Key
0
1       4321
2
3
4       1234
5       8765
6       9876
7
8       5678
9
```

```
int main(){
    int m=max;
    int employeekey[]={1234,5678,9876,43
                       8765};
for(int i=0;i<sizeof(employeekeys)/sizeof(empl
                    key[0]);(++){
        unsuitIntoHashtable(employeekeys[i],m
}
displayHashTable(m);
return 0;
}
```

O/p HashTable

Address	Key
0	
1	4321
3	
2	
4	1234
5	8765
6	9876
7	
8	5678
9	

#

Hash Table using remainder method

```
#include <stdio.h>
#define max 10
int hashTable[max]={0};
void insertIntoHashTable(int key, int m){
    int hashValue = key % m;
    while(hashTable[hashValue] != 0){
        hashValue = (hashValue+1) % m;
    }
    hashtable[hashValue]=key;
}
void displayHashTable(int m){
    printf("\n HashTable :\n");
    printf("Address\t key \n");
    for(int i=0; i< m; i++){
        printf("%d\t", i);
        if (hashTable[i] != 0){
            printf("%d", hashTable[i]);
        }
        printf("\n");
    }
}
```

LeetCode – I

Score of Parenthesis

```
int scoreOfParentheses(char* s) {
    int stack[50];
    int top = -1;
    int score = 0;

    for (int i = 0; i < strlen(s); i++) {
        if (s[i] == '(') {
            top++;
            stack[top] = 0;
        } else{
            if (stack[top] == 0) {
                stack[top] = 1;
            } else {
                int innerScore = 0;
                while (stack[top] != 0) {
                    innerScore += stack[top];
                    top--;
                }
                stack[top] = 2 * innerScore;
            }
        }
    }

    while (top != -1) {
        score += stack[top];
        top--;
    }

    return score;
}
```

19/02/24

Score of Parenthesis

```
int scoreofParentheses(char* s){
    int stack[50];
    int top = -1;
    int score = 0;
    for(int i=0; i<strlen(s); i++){
        if(s[i]=='('){
            top++;
            stack[top]=0;
        }else{
            if stack[top]==0){
                stack[top]=1;
            }else{
                int innerScore=0;
            while(stack[top]!=0){
                innerScore+=stack[top];
                top--;
            }
            stack[top]=2*innerScore;
        }}
    while(top!=-1){
        score+=stack[top];
        top--;
    }return score;
}
```

Delete the Middle Node of a Linked List

```
/**
 * Definition for singly-linked list.*/
#include <stdlib.h>
#include <stdio.h>
  struct ListNode {
      int val;
      struct ListNode *next;
  };

struct ListNode* deleteMiddle(struct ListNode* head) {
    if (head == NULL || head->next == NULL) {
        free(head);
        return NULL;
    }

    struct ListNode *ptr1 = head;
    struct ListNode *ptr2 = head;
    struct ListNode *ptr3 = head;
    int count = 0;
    while(ptr3!=NULL){
        count++;
        ptr3=ptr3->next;
    }
    int pos=0;
    while(pos<(count/2)){
        ptr2=ptr1;
        ptr1=ptr1->next;
        pos++;
    }

    ptr2->next=ptr1->next;
    free(ptr1);
    ptr1=NULL;
    return head;

    }
```

19/02/24 Delete The Middle Node of the linked list

```
struct ListNode* deleteMiddle(struct
ListNode* head){
    if(head==NULL || head->next==NULL){
        free(head);
        return NULL;
    }
struct ListNode *ptr1=head;
struct ListNode *ptr2=head;
struct ListNode *ptr3=head
int count=0;
while(ptr3!=NULL){
    count++;
    ptr3=ptr3->next;
}
int pos=0
while(pos<(count/2)){
    ptr2=ptr1;
    ptr1=ptr1->next;
    pos++;
}
ptr2->next=ptr1->next;
free(ptr1);
ptr1=NULL;
return head;
}
```

Odd Even Linked List

```
/*
  Definition for singly-linked list.*/
  #include <stdio.h>
  #include <stdlib.h>
  struct ListNode {
     int val;
     struct ListNode *next;
  };

struct ListNode* oddEvenList(struct ListNode* head) {
    if(head==NULL||head->next==NULL){
            return head;
    }
    struct ListNode *oddHead=head;
    struct ListNode *evenHead=head->next;
    struct ListNode *odd=oddHead;
    struct ListNode *even=evenHead;
    while(even !=NULL&&even->next!=NULL){
        odd->next=even->next;
        odd=odd->next;
        even->next=odd->next;
        even=even->next;
    }
    odd->next=evenHead;
    return oddHead;
}
```

19/02/24 Odd Even Linked List

```
struct ListNode* oddEvenList(struct
ListNode *head){
    if(head==NULL || head->next==NULL){
        return head;
    }
    struct ListNode *oddHead=head;
    struct ListNode *evenHead=head->next;
    struct ListNode *odd=oddHead;
    struct ListNode *even=evenHead;
    while(even!=NULL && even->next!=NULL){
        odd->next=even->next;
        odd=odd->next;
        even->next=odd->next;
        even=even->next;
    }
    odd->next=evenHead;
    return oddHead;
}
```

Leet Code – II

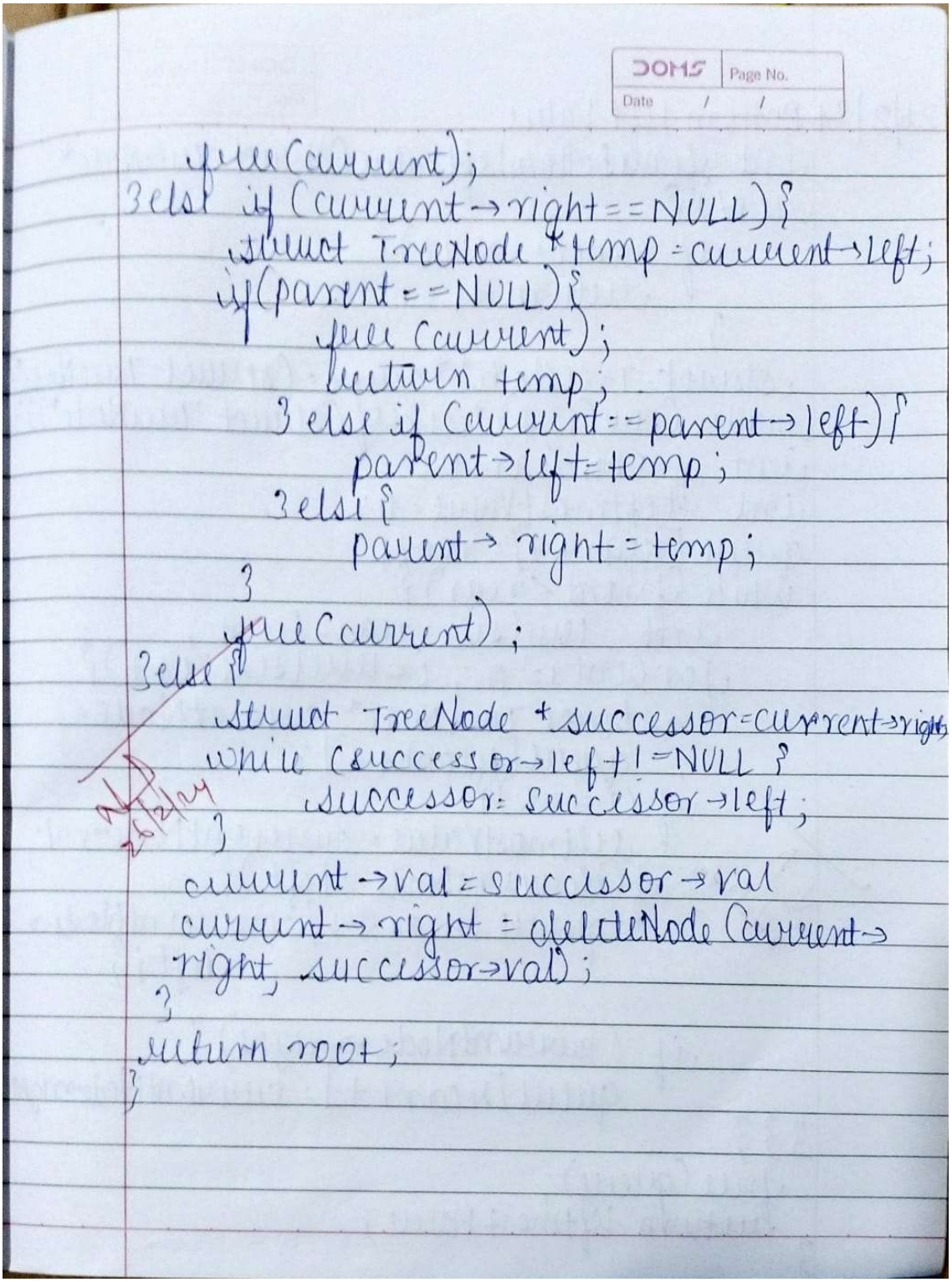
```
  #include <stdio.h>
  #include <stdlib.h>
  struct TreeNode {
      int val;
     struct TreeNode *left;
      struct TreeNode *right;
 };

struct TreeNode* deleteNode(struct TreeNode* root, int key) {
     struct TreeNode *current = root;
    struct TreeNode *parent;


    while (current != NULL && current->val != key) {
        parent = current;
        if (key < current->val) {
            current = current->left;
        } else {
            current = current->right;
        }
    }


    if (current == NULL) {
        return root;
    }

    if (current->left == NULL) {
        struct TreeNode *temp = current->right;
        if (parent == NULL) {
            free(current);
            return temp;
        } else if (current == parent->left) {
            parent->left = temp;
        } else {
            parent->right = temp;
        }
        free(current);
    } else if (current->right == NULL) {
        struct TreeNode *temp = current->left;
        if (parent == NULL) {
            free(current);
            return temp;
        } else if (current == parent->left) {
            parent->left = temp;
        } else {
            parent->right = temp;
        }
        free(current);
    } else {
```

```
        struct TreeNode *successor = current->right;
        while (successor->left != NULL) {
            successor = successor->left;
        }

        current->val = successor->val;


        current->right = deleteNode(current->right, successor->val);
    }

    return root;
}
```

26/02/24 LeetCode

Delete node from BST

```
struct TreeNode* deleteNode(struct
TreeNode* root, int key){
    struct TreeNode *current = root;
    struct TreeNode *parent = NULL;
while(current != NULL && current->val!=
{ parent = current;
  if (key < current->val){
     current = current->left
  } else {
     current = current->right;
  }
}
if (current == NULL){
   return root;
}
if (current->left == NULL){
    struct TreeNode *temp = current->right;
    if(parent == NULL){
        free(current);
        return temp;
    } else if (current == parent->left){
        parent->left = temp;
    } else {
        parent->right = temp;
```



```
free(current);
} else if (current->right == NULL) {
    struct TreeNode *temp = current->left;
    if(parent == NULL) {
        free(current);
        return temp;
    } else if (current == parent->left) {
        parent->left = temp;
    } else {
        parent->right = temp;
    }
    free(current);
} else {
    struct TreeNode *successor = current->right;
    while (successor->left != NULL) {
        successor = successor->left;
    }
    current->val = successor->val;
    current->right = deleteNode(current->right, successor->val);
}
return root;
}
```

Left Most Node

```
  #include <stdio.h>
  #include <stdlib.h>
  struct TreeNode {
      int val;
     struct TreeNode *left;
      struct TreeNode *right;
 };

int findBottomLeftValue(struct TreeNode* root) {
    if (root == NULL) {
        return -1;
    }

    struct TreeNode** queue = (struct TreeNode**)malloc(pow(10,4) *
sizeof(struct TreeNode*));
    int front = 0, rear = 0;
    int leftmostValue = 0;

    queue[rear++] = root;

    while (front < rear) {
        int levelSize = rear - front;

        for (int i = 0; i < levelSize; i++) {
            struct TreeNode* currentNode = queue[front++];

            if (i == 0) {
                leftmostValue = currentNode->val;
            }

            if (currentNode->left) {
                queue[rear++] = currentNode->left;
            }

            if (currentNode->right) {
                queue[rear++] = currentNode->right;
            }
        }
    }

    free(queue);
    return leftmostValue;
}
```

26/2/24 Bottom Left Value

```
int findBottomLeftValue(struct TreeNode* root){
    if (root==NULL){
        return -1;
    }
    struct TreeNode** queue = (struct TreeNode**)
    malloc(pow(10,4)*sizeof(struct TreeNode*));
    int front, rear=0;
    int leftmostValue=0;
    queue[rear++]=root;
    while(front<rear){
        int levelSize=rear-front;
        for(int i=0; i<levelSize; i++){
            struct TreeNode* currentNode=
                queue[front++];
            if(i==0){
                leftmostValue=currentNode->val;
            if(currentNode->left){
                queue[rear++]=currentNode->
                    left;}

            if(currentNode->right){
                queue[rear++]=currentNode->right
    }}}
    free(queue);
    return leftmostValue;
}
```