VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Data Structures using C Lab

(23CS3PCDST)

Submitted by

Sanchit Mehta (1BM23CS299)

in partial fulfilment 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
Sep-2024 to Jan-2025

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

CERTIFICATE

This is to certify that the Lab work entitled "Data Structures using C Lab (23CS3PCDST)" carried out by **Sanchit Mehta (1BM23CS299),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of Data Structures using C Lab (23CS3PCDST) work prescribed for the said degree.

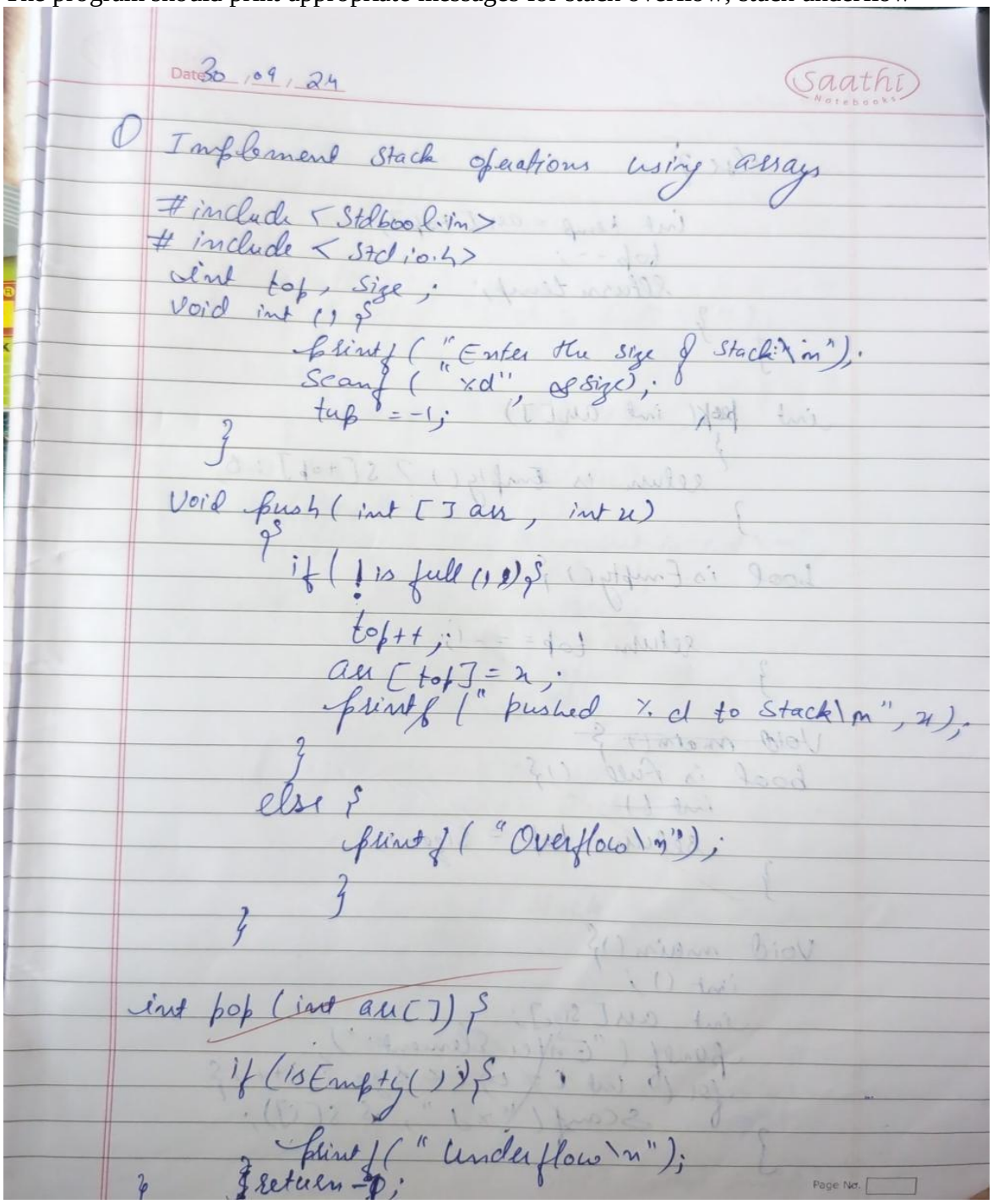
Geetha N Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor &HOD Department of CSE, BMSCE

Index

Github Link:
https://github.com/sanchit299/1BM23CS299_DS

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



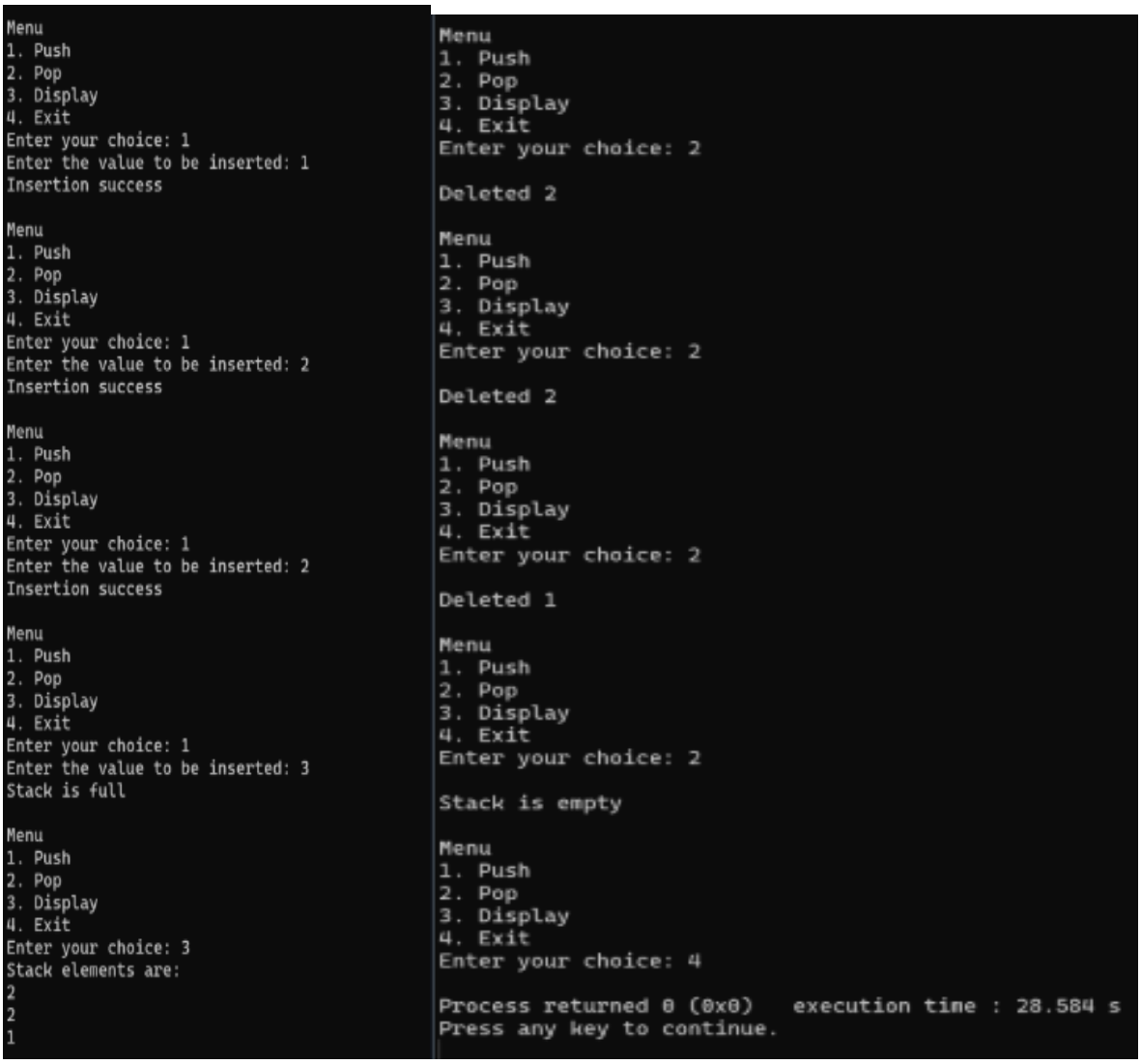
```
    else {
        int temp = arr[top];
        top--;
        return temp;
    }
}

int peek(int arr[])
{
    return isEmpty() ? S[top] : 0;
}

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

bool isFull() {
    return top = size-1;
}

void main() {
    int i;
    int arr[size];
    printf("Enter Element:");
    for(int i=0; i<size; i++) {
        scanf("%d", &S[i]);
    }
```

```
push(arr, 10);
push(arr, 20);
push(arr, 30);

if(isfull()) {
    printf("Stack is full\n");
}
else {
    printf("Not full\n");
}
int topElem = peek(arr);
if(topElem != -1) {
    printf("Top element is: %d\n",
            topElem);
int poppedVal = pop(arr);
if(poppedVal != -1) {
    printf("Popped %d from stack\n",
        poppedVal);
}
if(isEmpty()) {
    printf("Stack is empty\n");
}
else {
    printf("Stack is not empty\n");
}
pop(arr);
pop(arr);
if(isEmpty()) {
    printf("Stack is empty\n");
}
else {
    printf("Stack is not empty\n");
}
```

Code:

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 3

void push(int value);
void pop();
void display();

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

int main() {
   int value, choice;

   while(1) {
      printf("\nMenu\n");
      printf("1. Push\n");
      printf("2. Pop\n");
      printf("3. Display\n");
      printf("4. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);  // Corrected the format specifier

      switch(choice) {
         case 1:
            printf("Enter the value to be inserted: ");
            scanf("%d", &value);
            push(value);
            break;

         case 2:
            pop();
            break;

         case 3:
            display();
            break;

         case 4:
            exit(0);

         default:
            printf("Invalid choice\n");
      }
   }
```

```
    return 0;
}

void push(int value) {
    if(top == SIZE - 1) {
        printf("Stack is full\n");
    } else {
        top++;
        stack[top] = value;
        printf("Insertion success\n");
    }
}

void pop() {
    if(top == -1) {
        printf("\nStack is empty\n");
    } else {
        printf("\nDeleted %d\n", stack[top]);
        top--;
    }
}

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



2) WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide)

Date 4/10/24

Q WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and binary operators +, -, * and /

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 100
int prec (char c) {
    if (c == '^')
        return 3;
    else if (c == '/' || c == '*')
        return 2;
    else if (c == '+' || c == '-')
        return 1;
    else
        return -1; }
```

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

void push (char x){
    if (top == MAX-1){
        printf("Stack overflow\n");
        return;
    }
    top++;
    stack[top] = x;
}

char pop(){
    if (top == -1){
        printf("Stack underflow\n");
        return -1;
    }
    char ans = stack[top];
    top--;
    return ans;
}

int isOperator (char x){
    return (x == '+' || x == '-' || x == '*' ||
            x == '/' || x == '^');
}
```

```
Void infixToPostfix (char exp[]) {
    char x;
    for (int i=0; exp[i] != '\0'; i++) {
        if (isalnum(exp[i])) {
            printf("%c", exp[i]);
        }
        else if (exp[i] == '(') {
            push(exp[i]);
        }
        else if (exp[i] == ')') {
            ~~while (exp[i] != '(')~~
            while ((x = pop()) != '(')
                printf("%c", x);
        }
        else if (isOperator(exp[i])) {
            while (top != -1 &&
                precedence(stack[top]) >=
                    precedence(exp[i])) {
                printf("%c", pop());
            }
            push(exp[i]);
        }
    }
```

```
    while (top != -1) {
        printf("%c", pop());
    }
}
Void main() {
    char exp[MAX];
    printf("Enter infix expression: ");
    scanf("%s", exp);
    printf("postfix expression: ");
    infixToPostfix(exp);
}
```

seen
op seen
GK
14/10/24

Output :- Enter infix expression : ((P+Q)*R^Q^T/S)
Postfix expression: PQ+RQ^T^*S/

Code:

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

#define MAX 3

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

int is_empty() {
    return front == -1 || front > rear;
}

int is_full() {
    return rear == MAX - 1;
}

void insert(int value) {
    if (is_full()) {
        printf("Queue overflow! Cannot insert %d\n", value);
    } else {
        if (front == -1) {
            front = 0;
        }
        queue[++rear] = value;
        printf("Inserted %d\n", value);
    }
}

void delete() {
    if (is_empty()) {
        printf("Queue underflow! Cannot delete\n");
```

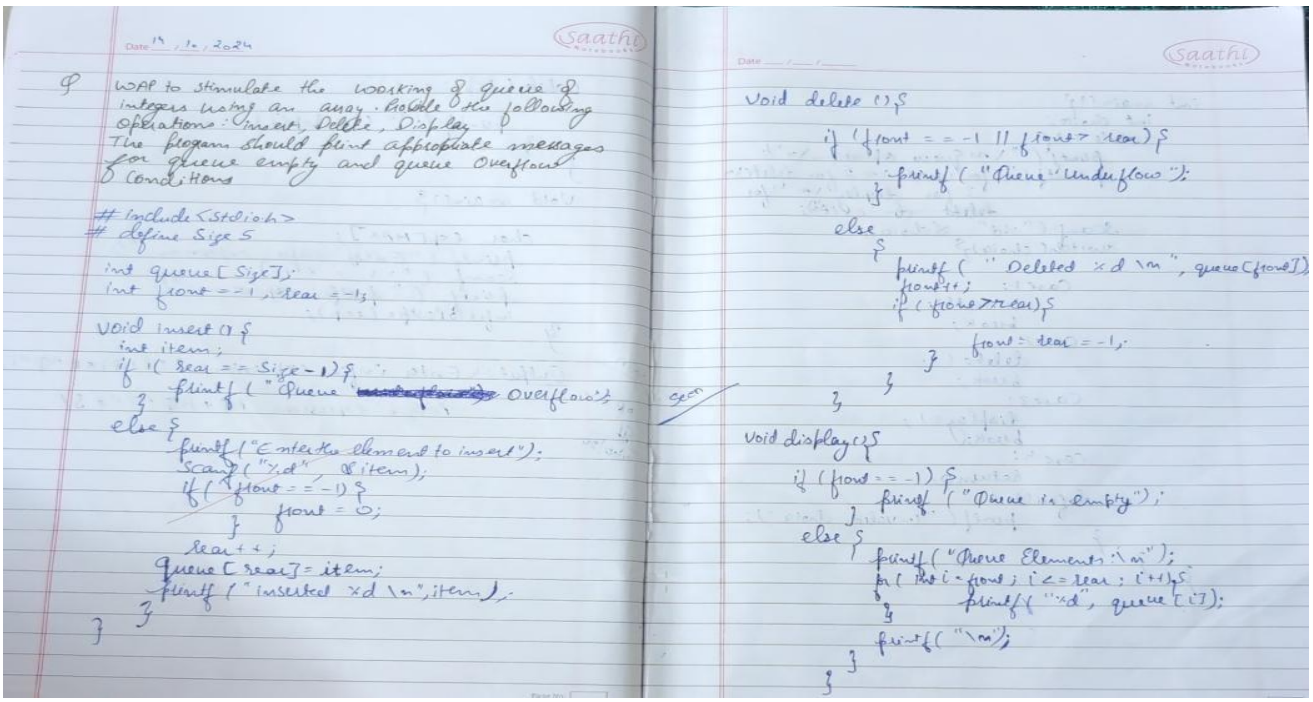
```
    } else {
        printf("Deleted %d\n", queue[front]);
        front++;
        if (is_empty()) {
            front = rear = -1;
        }
    }
}

void display() {
    if (is_empty()) {
        printf("Queue is empty\n");
    } else {
        printf("Queue: ");
        for (int i = front; i <= rear; i++) {
            printf("%d ", queue[i]);
        }
        printf("\n");
    }
}

int main() {
    int choice, value;

    while (1) {
        printf("\nQueue Operations:\n");
        printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
```

```
            case 1:
                printf("Enter a value to insert: ");
                scanf("%d", &value);
                insert(value);
                break;
            case 2:
                delete();
                break;
            case 3:
                display();
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }
    return 0;
}
```

```
Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter a value to insert: 1
Inserted 1

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter a value to insert: 2
Inserted 2

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter a value to insert: 3
Inserted 3

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter a value to insert: 4
Queue overflow! Cannot insert 4

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue: 1 2 3
```

```
Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted 1

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted 2

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted 3

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Queue underflow! Cannot delete

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 7
Invalid choice! Please try again.
```

3) a) WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display The program should print appropriate messages for queue empty and queue overflow conditions



```
int main();
    int choice;
    while (1){
        printf("\n Queue Operations\n");
        printf("1 for insert \n 2 for delete\n
                3 for display \n 4 for
                delete to exit");
        scanf("%d", &choice);
        switch(choice){

            case 1:
                insert();
                break;
            case2:
                delete();
                break;
            case3:
                display();
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice");
        }
    }
}
```

Output:→

```
Queue Operations
1 for insert
2 for delete
3 for display
4 for exit

enter the element to be inserted
1
inserted 1

Queue Operation
1 for insert
2 for delete
3 for display
4 for exit

enter the element to be inserted
2
inserted 2

enter the element to be inserted
3
inserted 3

Enter the element to be inserted
4
inserted 4
enter the element to be inserted
5
inserted 5
queue overflow
```


```
int main();
    int choice;
    while (1){
        printf("\n Queue Operations\n");
        printf("1 for insert \n 2 for delete\n
                3 for display \n 4 for
                delete to exit");
        scanf("%d", &choice);
        switch(choice){

            case 1:
                insert();
                break;
            case2:
                delete();
                break;
            case3:
                display();
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice");
        }
    }
}
```

Output:→

```
Queue Operations
1 for insert
2 for delete
3 for display
4 for exit

enter the element to be inserted
1
inserted 1

Queue Operation
1 for insert
2 for delete
3 for display
4 for exit

enter the element to be inserted
2
inserted 2

enter the element to be inserted
3
inserted 3

Enter the element to be inserted
4
inserted 4
enter the element to be inserted
5
inserted 5
queue overflow
```

Date ___/___/_____

Queue operations

1 insert
2 for delete
3 for display
4 for exit

3

Queue elements 1 2 3 4 5

Queue operations

1 for insert
2 for delete
3 for display
4 for exit

2

Queue operations

1 for insert
2 for delete
3 for display
4 for exit

Seen
GR
14/10/24

3

Queue elements 2 3 4 5

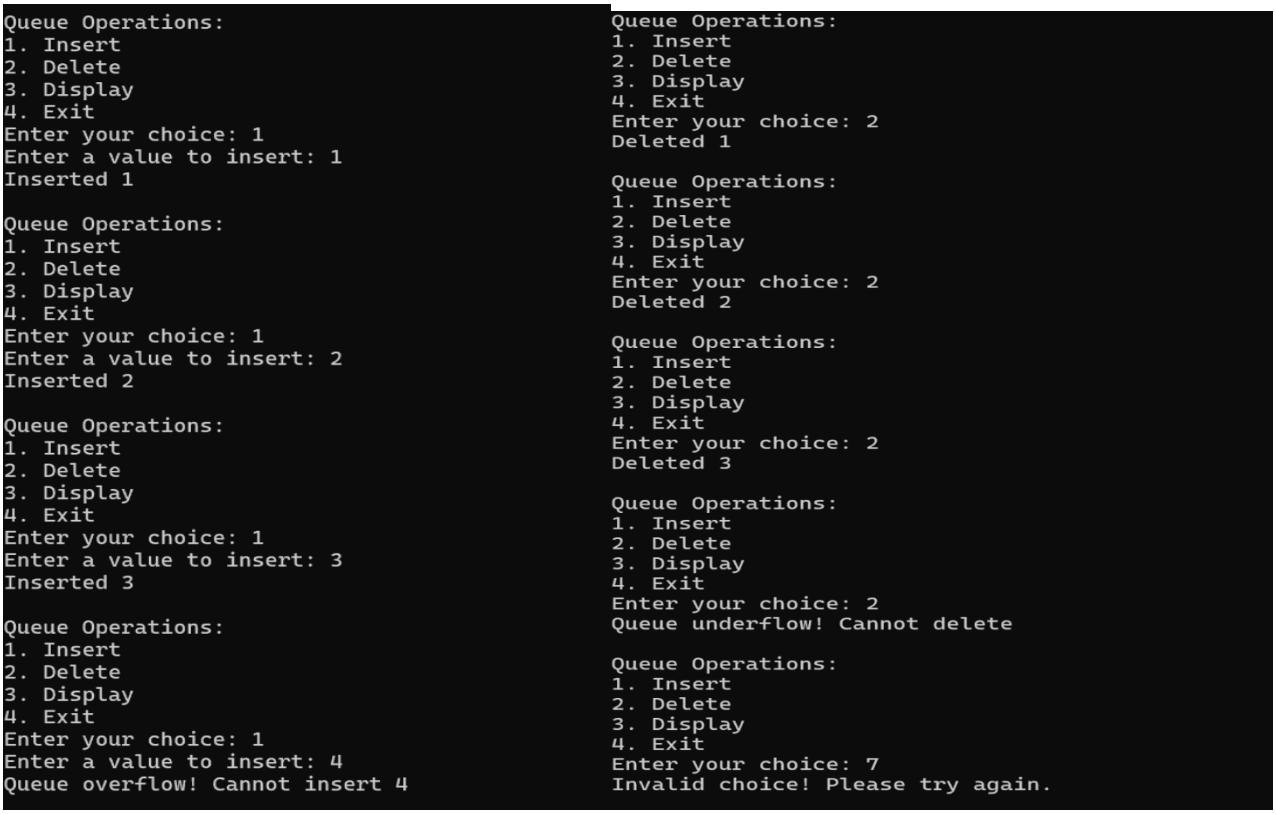
Code:

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

#define MAX 3

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

int is_empty() {
    return front == -1 || front > rear;
}

int is_full() {
    return rear == MAX - 1;
}

void insert(int value) {
    if (is_full()) {
        printf("Queue overflow! Cannot insert %d\n", value);
    } else {
        if (front == -1) {
            front = 0;
        }
```

```
        queue[++rear] = value;
        printf("Inserted %d\n", value);
    }
}

void delete() {
    if (is_empty()) {
        printf("Queue underflow! Cannot delete\n");
    } else {
        printf("Deleted %d\n", queue[front]);
        front++;
        if (is_empty()) {
            front = rear = -1;
        }
    }
}

void display() {
    if (is_empty()) {
        printf("Queue is empty\n");
    } else {
        printf("Queue: ");
        for (int i = front; i <= rear; i++) {
            printf("%d ", queue[i]);
```

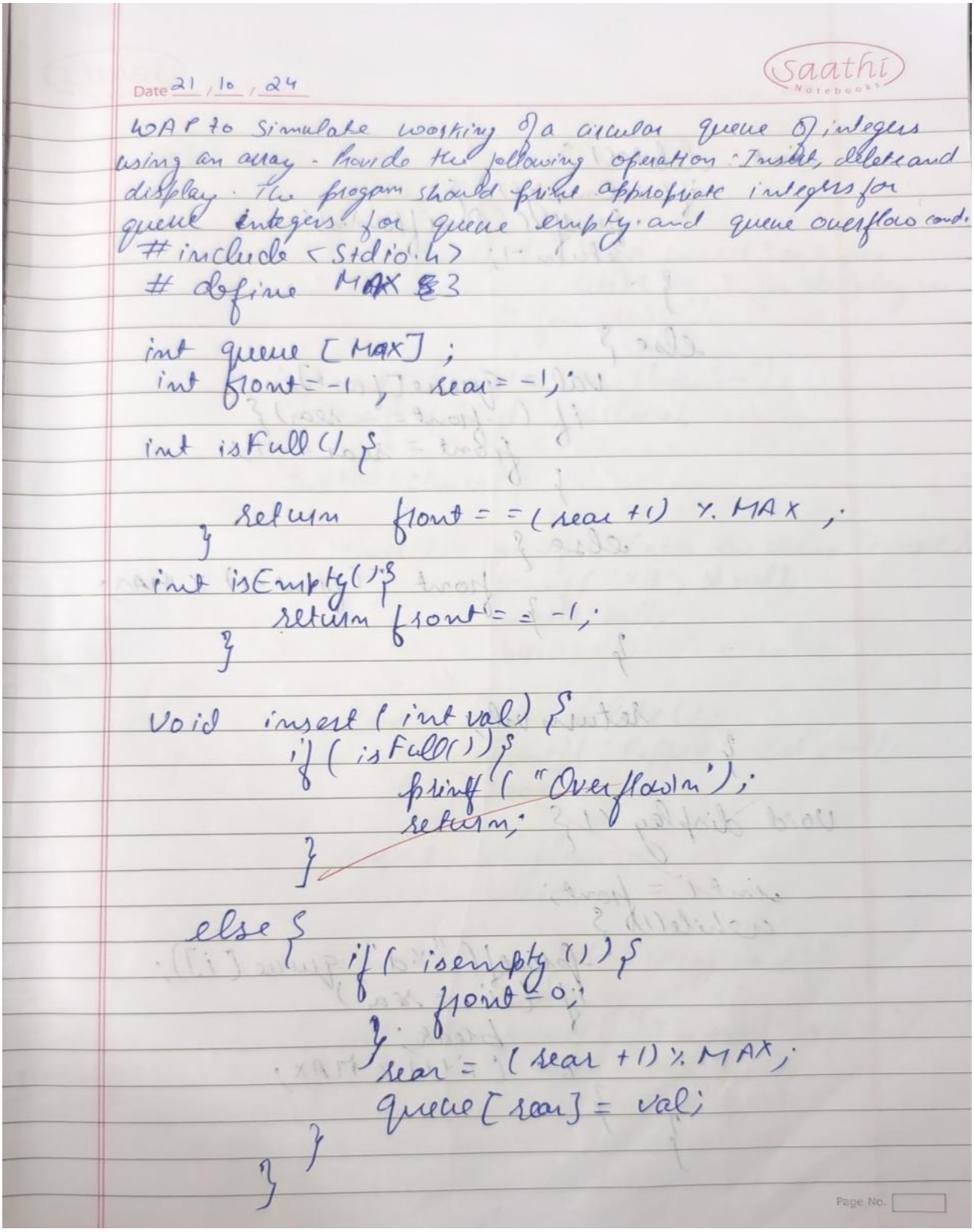
```
        }
        printf("\n");
    }
}

int main() {
    int choice, value;

    while (1) {
        printf("\nQueue Operations:\n");
        printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter a value to insert: ");
                scanf("%d", &value);
                insert(value);
                break;
            case 2:
                delete();
                break;
            case 3:
```

```
            display();
            break;
        case 4:
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n")}
    return 0;
}
```



3b) WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & Display The program should print appropriate messages for queue empty and queue

overflow conditions



Date 21/10/24

Saathi Notebooks

WAP to simulate working of a circular queue of integers using an array. Have do the following operation: Insert, delete and display. The program should print appropriate integers for queue integers for queue empty and queue overflow cond.

```
#include <stdio.h>
# define MAX 3

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

int isFull(){
    return front == (rear+1) % MAX;
}
int isEmpty(){
    return front == -1;
}

Void insert (int val){
    if (isFull()){
        printf ("Overflow\n");
        return;
    }
else {
    if (isempty()){
        front = 0;
    }
    rear = (rear+1) % MAX;
    queue[rear] = val;
}
}
```

Page No.

```
int delete() {
    if (isEmpty()) {
        printf("Overflow\n");
        return -1;
    }
    else {
        val = queue[front];
        if (front == rear) {
            front = rear = -1;
        }
        else {
            front = (front+1) % MAX;
        }
        return val;
    }
}
void display() {
    int i = front;
    while(1) {
        printf("%d", queue[i]);
        if (i == rear)
            break;
        i = (i+1) % MAX;
    }
}
```

```
void main() {
    int choice, val;
    while(1) {
        printf("1 for insert \n 2 for delete \n 3 for display \n 4 for Exit \n");
        printf("Enter the choice \n");
        scanf("%d", &choice);
        switch(choice) {
            case 1: printf("Enter the value to insert");
                    scanf("%d", &val);
                    insert(val);
                    break;
            case 2: val = delete();
                    printf("Popped = %d", val);
                    break;
            case 3: display();
                    break;
            case 4: printf("Exiting \n");
                    return;
            default: printf("Invalid choice \n");
        }
    }
}
```

Seen
execute

output:→ 1 for insert
2 for delete
3 for display
4 for exit

Enter the choice: 1
Enter the value to be inserted 1

Enter the choice: 1
Enter the value to be inserted: 2
Enter the choice: 1
Enter the value to be inserted: 3
Enter the choice: 1
Queue Overflow

Enter the choice: 2
Popped = 1

Enter the choice: 2
popped = 2

Enter the choice: 1
Enter the value to be inserted: 8

Enter the choice: 1
Enter the value to be inserted: 7

Seen

Enter the choice: 3
3 8 7

Enter the choice: 8
Invalid choice

Enter the choice: 2
popped = 7

Enter the choice: 2
popped = 8

Enter the choice: 2
popped = 7

Enter the choice: 2
Underflow

Code:

```
#include <stdio.h>
#define MAX 3

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

int isfull() {
   return (rear + 1) % MAX == front;
}

int isempty() {
   return front == -1;
}

void insert(int val) {
   if (isfull()) {
      printf("Overflow\n");
      return;
   }
   if (isempty()) {
      front = 0;  // Initialize front
   }
   rear = (rear + 1) % MAX;
   queue[rear] = val;
}

int delete() {
   if (isempty()) {
      printf("Underflow\n");
      return -1;
   }
   int val = queue[front];
   if (front == rear) {
      front = rear = -1;  // Queue is now empty
   } else {
      front = (front + 1) % MAX;
   }
   return val;
}
```

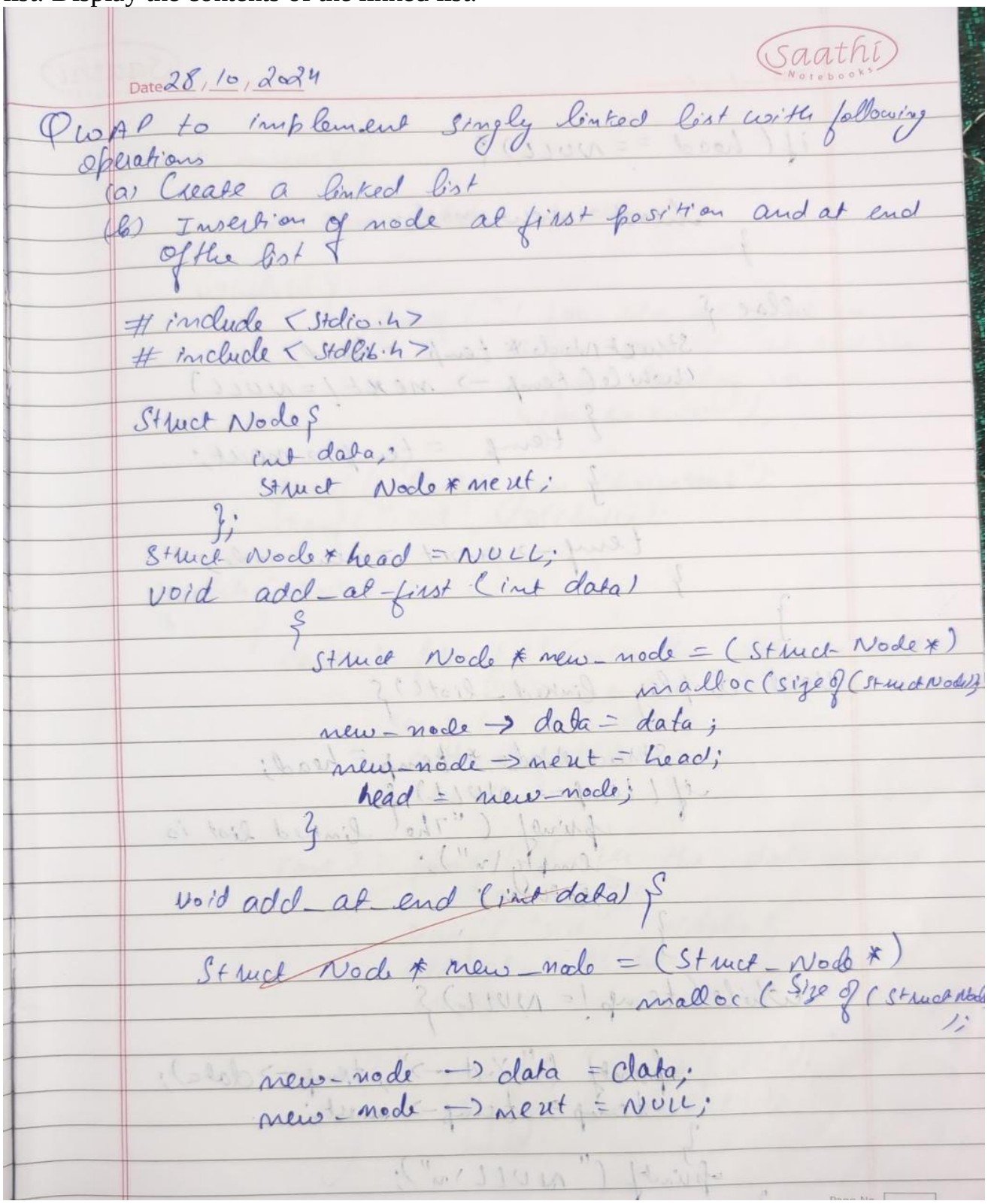
```
void display() {
    if (isempty()) {
        printf("Queue is empty\n");
        return;
    }
    int i = front;
    while (1) {
        printf("%d ", queue[i]);
        if (i == rear) {
            break;
        }
        i = (i + 1) % MAX;
    }
    printf("\n");  // Newline for better readability
}

int main() {
    int choice, val;
    while (1) {
        printf("1 for insert\n2 for delete\n3 for display\n4 for exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter the value to insert: ");
                scanf("%d", &val);
                insert(val);
                break;
            case 2:
                val = delete();
                if (val != -1) { // Check for underflow before printing
                    printf("Popped = %d\n", val);
                }
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting...\n");
```

```
            return 0;
        default:
            printf("Invalid choice\n");
    }
  }
}
```

```
1 for insert
2 for delete
3 for display
4 for exit
Enter your choice: 1
Enter the value to insert: 1
1 for insert
2 for delete
3 for display
4 for exit
Enter your choice: 1
Enter the value to insert: 2
1 for insert
2 for delete
3 for display
4 for exit
Enter your choice: 1
Enter the value to insert: 3
1 for insert
2 for delete
3 for display
4 for exit
Enter your choice: 1
Enter the value to insert: 4
Overflow
1 for insert
2 for delete
3 for display
4 for exit
Enter your choice: 2
Popped = 1
1 for insert
2 for delete
3 for display
4 for exit
Enter your choice: 2
Popped = 2
1 for insert
2 for delete
3 for display
4 for exit
Enter your choice: 2
Popped = 3
```

```
Popped = 3
1 for insert
2 for delete
3 for display
4 for exit
Enter your choice: 2
Underflow
1 for insert
2 for delete
3 for display
4 for exit
Enter your choice: 1
Enter the value to insert: 5
1 for insert
2 for delete
3 for display
4 for exit
Enter your choice: 1
Enter the value to insert: 7
1 for insert
2 for delete
3 for display
4 for exit
Enter your choice: 3
5 7
1 for insert
2 for delete
3 for display
4 for exit
Enter your choice: 8
Invalid choice
1 for insert
2 for delete
3 for display
4 for exit
Enter your choice: 4
Exiting...

Process returned 0 (0x0)   execution time : 118.059 s
Press any key to continue.
```

4)WAP to Implement Singly Linked List with following operations a) Createalinkedlist. b) Insertion of a node at first position, at any position and at end of list. Display the contents of the linked list.



Date 28/10/2024

Q WAP to implement singly linked list with following operations
(a) Create a linked list
(b) Insertion of node at first position and at end of the list

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

struct Node{
    int data;
    struct Node *next;
};
struct Node *head = NULL;
void add_at_first (int data)
{
    struct Node *new_node = (struct Node *)
                malloc(sizeof(struct Node));
    new_node -> data = data;
    new_node -> next = head;
    head = new_node;
}

void add_at_end (int data){
    struct Node *new_node = (struct Node *)
                malloc (sizeof(struct Node
                ));

    new_node -> data = data;
    new_node -> next = NULL;
```

```
if (head == NULL) {
    head = new_node;
}
else {
    struct Node * temp = head;
    while (temp -> next != NULL)
    { temp = temp -> next;
    }
    temp -> next = new_node;
}
}
void display_linked_list() {
    Struct Node * temp = head;
    if (temp == NULL) {
        printf ("The linked list is
        empty \n");
        return;
    }
while (temp != NULL) {
    printf ("%d ->", temp -> data);
    temp = temp -> next;
}
printf ("NULL \n");
}
```

```
int main() {
    int choice, data, position;
while (1) {
    printf ("1 for add at first \n
             2 for add at end \n
             3 for display \n
             4 for exit. \n");
printf ("Enter your choice:");
scanf ("%d", &choice);
switch (choice)
{
    case 1: printf ("Enter the data to add
            at first:");
        scanf ("%d", &data
        add_at_first(data);
        break;
    case 2: printf ("Enter the data to add
            at end:");
        scanf ("%d", &data);
        add_at_end(data);
        break;
    case 3: printf ("Linked list:");
        display_linked_list();
        break;
```

```
        case 4: printf ("Exiting\n");
                exit(0);

        default:
                printf ("Invalid choice\n");
        }
    }
    Return 0;
}
```

Seen

Output :-> 1 for add at first 2 for add at end 3 for display 4 for exit
Enter your choice: 1
Enter data to add at first: 1
Enter your choice: 2
Enter data to add at end: 2
Enter your choice: 3
1 -> 2 ->
Enter your choice: 1
Enter data to add at first: 7
Enter your choice: 3
3 -> 1 -> 2 ->

Seen

Code:

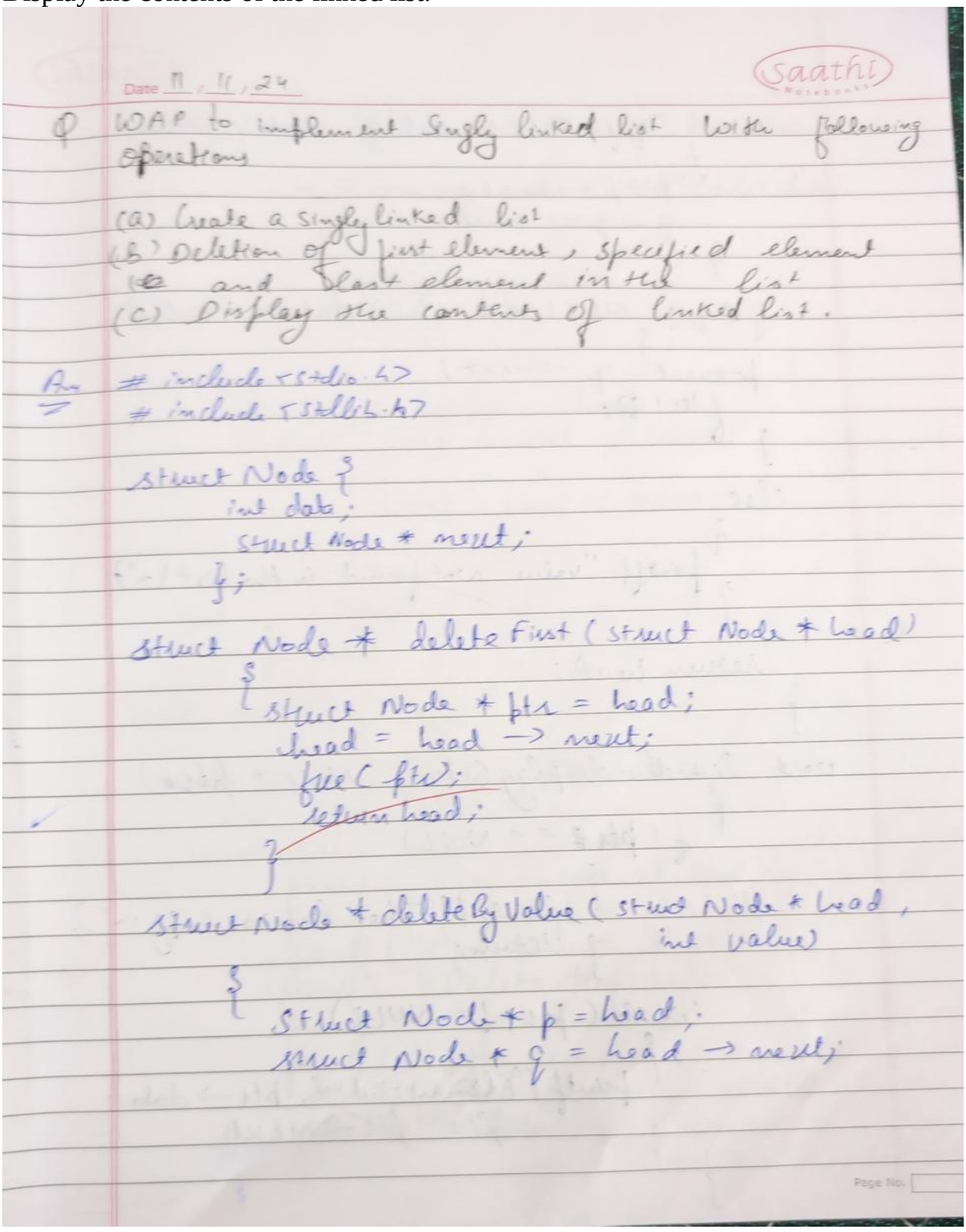
```
#include <stdio.h>
#include <stdlib.h>

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

struct Node* head = NULL;

void create_linked_list(int data_list[], int n) {
    for (int i = 0; i < n; i++) {
        insert_at_end(data_list[i]);
    }
}

void insert_at_beginning(int data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = data;
    new_node->next = head;
    head = new_node;
}

void insert_at_end(int data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = data;
    new_node->next = NULL;

    if (head == NULL) {
        head = new_node;
        return;
    }

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

```
}

void display_linked_list() {
    struct Node* temp = head;
    if (temp == NULL) {
        printf("The linked list is empty.\n");
        return;
    }
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
    int choice, data;
    int data_list[] = {10, 20, 30};
    int n = sizeof(data_list) / sizeof(data_list[0]);

    create_linked_list(data_list, n);

    while (1) {
        printf("\nChoose an operation:\n");
        printf("1. Display linked list\n");
        printf("2. Insert at beginning\n");
        printf("3. Insert at end\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                display_linked_list();
                break;
            case 2:
                printf("Enter a value to insert at the beginning: ");
                scanf("%d", &data);
                insert_at_beginning(data);
                break;
```

```
            case 3:
                printf("Enter a value to insert at the end: ");
                scanf("%d", &data);
                insert_at_end(data);
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}
```

```
Choose an operation:
1. Display linked list
2. Insert at beginning
3. Insert at end
4. Exit
Enter your choice: 2
Enter a value to insert at the beginning: 1

Choose an operation:
1. Display linked list
2. Insert at beginning
3. Insert at end
4. Exit
Enter your choice: 2
Enter a value to insert at the beginning: 1

Choose an operation:
1. Display linked list
2. Insert at beginning
3. Insert at end
4. Exit
Enter your choice: 3
Enter a value to insert at the end: 3

Choose an operation:
1. Display linked list
2. Insert at beginning
3. Insert at end
4. Exit
Enter your choice: 1
1 -> 1 -> 10 -> 20 -> 30 -> 3 -> NULL

Choose an operation:
1. Display linked list
2. Insert at beginning
3. Insert at end
4. Exit
Enter your choice: 4

Process returned 0 (0x0)   execution time : 41.926 s
Press any key to continue.
```

5)WAP to Implement Singly Linked List with following operations a) Create a linked list. b) Deletion of first element, specified element and last element in the list. c) Display the contents of the linked list.



Date 11/11/24

Saathi Notebooks

Q WAP to implement Singly linked list with following operations

(a) Create a singly linked list
(b) Deletion of first element, specified element and last element in the list
(c) Display the contents of linked list.

Ans

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

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

struct Node * deleteFirst(struct Node * head)
{
    struct Node * ptr = head;
    head = head -> next;
    free(ptr);
    return head;
}

struct Node * deleteByValue(struct Node * head,
                            int value)
{
    struct Node * p = head;
    struct Node * q = head -> next;
```

Page No.

```
while (q != NULL && q->data != value)
    {
      p = p->next;
      q = q->next;
    }
if (q != NULL && q->data == value)
  {
  p->next = q->next;
  free(q);
  }
else
  {
  printf("value not found in the list\n");
  }
return head;
}
void linkedlistdisplay(struct Node *ptr)
{
  if (ptr == NULL)
      {
        printf("The list is empty\n");
      return;
      }
  while (ptr != NULL)
    {
      printf("Element: %d\n", ptr->data);
        ptr = ptr->next;
    }
}
```

```
Void create (struct Node ** head, int data)
{
  struct Node * newNode = createNode(data)
  if (*head == NULL)
    {
      *head = newNode;
      return;
    }

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

      temp->next = newNode;
  }

Void main() {
  Struct Node *head = NULL;
  int choice, val;
  while (1) {
    printf("1 for Insert at end");
    printf("2 for Delete first");
    printf("3 for Delete last");
    printf("4 Delete specific value");
    printf("5 for display");
    printf("0 for exit");
    printf("Enter your choice:");
    scanf("%d", &choice);
```

```
Switch (choice) {

  Case 1: printf("Enter the value to insert:");
          scanf("%d", &val);
          create(&head, val);
          break;

  Case 2: head = deleteFirst(head);
          break;

  Case 3: head = deleteLast(head);
          break;

  Case 4: printf("Enter the value to delete:");
          scanf("%d", &value);
          head = deleteByValue(head, value);
          break;

  Case 5: linkedlistdisplay(head);
          break;

  Case 6: printf("Exit");
          return 0;

  default: printf("Invalid choice\n");
           break;

    }
}
```

```
Struct Node * deleteAtLast(Struct Node *head)
  {
  if (head == Null) {
      return head;
    }

  Struct Node *p = head;
  Struct Node *q = head->next;

  if (q == NULL) {
      free(p);
      return NULL;
    }

  while (q->next != NULL)
      {
        p = p->next;
        q = q->next;
      }

    p->next = NULL
    free(q);
    Return head;
}
```

Date __/__/____

O/P
1: Insert at End
2: Delete at first
3: Delete at last
4: Delete By Value
5: Display
6: Exit

Enter your choice: 1
Enter the value to insert: 2

Enter your choice: 1
Enter the Value to insert: 56

Enter your choice: 1
Enter the value to insert: 89

Enter your choice: 4
Enter value to delete: 56

Enter your choice: 5
Element: 2
Element: 89

Enter your choice: 2
Element: 89

Seen 16/12/24

Enter your choice: 1
Enter the value to insert: 67

Enter your choice: 3
Enter your choice: 5
Element: 89

Code:

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

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

struct Node *deleteFirst(struct Node *head) {
    if (head == NULL) return NULL;
    struct Node *ptr = head;
    head = head->next;
    free(ptr);
    return head;
}

struct Node *deleteByValue(struct Node *head, int value) {
    if (head == NULL) return NULL;
    struct Node *p = head;
    struct Node *q = head->next;

    if (head->data == value) {
        head = deleteFirst(head);
        return head;
    }

    while (q != NULL && q->data != value) {
        p = p->next;
        q = q->next;
    }

    if (q != NULL && q->data == value) {
        p->next = q->next;
        free(q);
    } else {
        printf("Value not found\n");
    }

    return head;
```

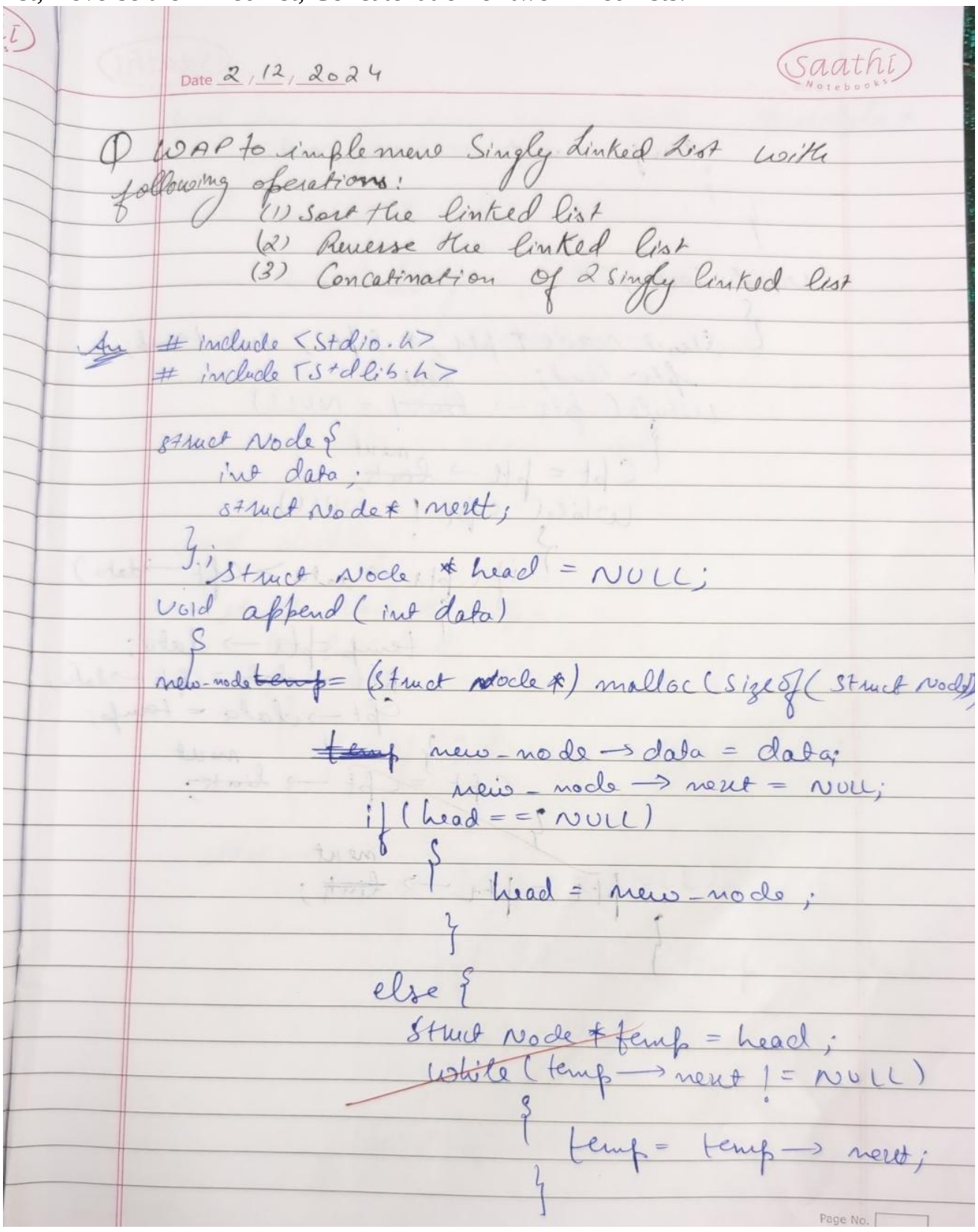
```
}

void create(struct Node **head, int data) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed!\n");
        return;
    }
    newNode->data = data;
    newNode->next = NULL;

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

struct Node *deleteAtLast(struct Node *head) {
    if (head == NULL) return NULL;

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

    struct Node *p = head;
    struct Node *q = head->next;
    while (q->next != NULL) {
        p = p->next;
        q = q->next;
    }

    p->next = NULL;
    free(q);
    return head;
```

```
}

void linkedListDisplay(struct Node *ptr) {
   if (ptr == NULL) {
      printf("The list is empty.\n");
      return;
   }

   while (ptr != NULL) {
      printf("Element: %d\n", ptr->data);
      ptr = ptr->next;
   }
}

int main() {
   struct Node *head = NULL;
   int choice, val;
   while (1) {
      printf("1: Insert at end\n");
      printf("2: Delete at first\n");
      printf("3: Delete at last\n");
      printf("4: Delete by value\n");
      printf("5: Display\n");
      printf("6: Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch (choice) {
         case 1:
            printf("Enter the value to insert: ");
            scanf("%d", &val);
            create(&head, val);
            break;
         case 2:
            head = deleteFirst(head);
            break;
         case 3:
            head = deleteAtLast(head);
            break;
         case 4:
```

```
            printf("Enter the value to delete: ");
            scanf("%d", &val);
            head = deleteByValue(head, val);
            break;
        case 5:
            linkedListDisplay(head);
            break;
        case 6:
            printf("Exiting...\n");
            return 0;
        default:
            printf("Invalid choice\n");
      }
   }
   return 0;
}
```

```
1: Insert at end
2: Delete at first
3: Delete at last
4: Delete by value
5: Display
6: Exit
Enter your choice: 1
Enter the value to insert: 1
1: Insert at end
2: Delete at first
3: Delete at last
4: Delete by value
5: Display
6: Exit
Enter your choice: 1
Enter the value to insert: 2
1: Insert at end
2: Delete at first
3: Delete at last
4: Delete by value
5: Display
6: Exit
Enter your choice: 5
Element: 1
Element: 2
1: Insert at end
2: Delete at first
3: Delete at last
4: Delete by value
5: Display
6: Exit
Enter your choice: 2
1: Insert at end
2: Delete at first
3: Delete at last
4: Delete by value
5: Display
6: Exit
Enter your choice: 5
Element: 2
1: Insert at end
2: Delete at first
3: Delete at last
4: Delete by value
5: Display
6: Exit
Enter your choice: 1
Enter the value to insert: 6
1: Insert at end
2: Delete at first
3: Delete at last
4: Delete by value
5: Display
6: Exit
Enter your choice: 4
Enter the value to delete: 6
1: Insert at end
2: Delete at first
3: Delete at last
4: Delete by value
5: Display
6: Exit
Enter your choice: 5
Element: 2
1: Insert at end
2: Delete at first
3: Delete at last
4: Delete by value
5: Display
6: Exit
Enter your choice: 6
Exiting...

Process returned 0 (0x0)   execution time : 88.490 s
Press any key to continue.
```

6) a) WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists.



Date 2/12/2024

Saathi Notebooks

Q WAP to implement Singly Linked List with following operations:
(1) Sort the linked list
(2) Reverse the linked list
(3) Concatination of 2 singly linked list

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

struct Node{
    int data;
    struct Node* next;
};
struct Node *head = NULL;
void append(int data)
{
new_node= (struct Node*) malloc(sizeof(struct Node));
    new_node->data = data;
    new_node->next = NULL;
    if(head == NULL)
    {
        head = new_node;
    }
    else {
        struct Node *temp = head;
        while(temp->next != NULL)
        {
            temp = temp->next;
        }
```

Page No.

```
        temp -> next = new_node;
    }
}
void Sorting()
{
    struct node *ptr, *cpt, int temp;
    ptr = head;
    while( ptr -> next != NULL)
    {
        cpt = ptr -> next;
        while( cpt != NULL)
        {
            if( ptr -> data > cpt -> data)
            {
                temp = ptr -> data;
                ptr -> data = cpt -> data;
                cpt -> data = temp;
            }
            cpt = cpt -> next;
        }
        ptr = ptr -> next;
    }
```

```
void concat ( struct Node * head1 , struct Node * head2).
{
    struct Node * ptr = NULL;
        ptr = head1;
    while (ptr -> next != NULL)
        {
            ptr = ptr -> next;
        }
    ptr -> next = head2;
}
void display()
{
    struct Node * temp;
        temp = head;
    if( temp != NULL)
    {
        printf(" List is empty");
    }
    else
    {
        while ( temp != NULL)
        {
            printf(" %d ->", temp -> data);
            temp = temp -> next;
        }
        printf("\n");
    }
```

Ans(6a)

```
void reverse ( struct node ** head)
{
    struct node *prev = NULL;
    struct node * curr = *head;
    struct node * next = NULL;
    while ( curr != NULL) {
        next = curr -> next;
        curr -> next = prev;
        prev = curr;
        curr = next;
    }
    *head = prev;
}
```

Seen 21/12/24

Saathi Notebooks

Date __/__/____

```
int main()
{
    struct node *list1 = NULL;
    struct node *list2 = NULL;
    int choice, data;
    while(1){
        printf(" 1 for Append to list1 \n
2 for append to list2 \n 3 concatination of
list2 to list1 \n 4 for display list1 \n
5 for display list2 \n 6 for exit");

    printf("Enter your choice:");
    scanf("%d", &choice);

    switch(choice)
    {
    case 1:
        printf("Add to list1 :");
        scanf("%d", &data);
        append(&list1, data);
        break;
    case 2:
        printf("Add to list2 :");
        scanf("%d", &data);
        append(&list2, data);
        break;

    case 3: concat(&list1, list2);
        break;

    case 4: printf("list1 :");
        display(list1);
        break;
```

```
int main()
{
    struct node *list1 = NULL;
    struct node *list2 = NULL;
    int choice, data;
    while(1){
        printf(" 1 for Append to list1 \n
2 for append to list2 \n 3 concatination of
list2 to list1 \n 4 for display list1 \n
5 for display list2 \n 6 for exit");

    printf("Enter your choice:");
    scanf("%d", &choice);

    switch(choice)
    {
    case 1:
        printf("Add to list1 :");
        scanf("%d", &data);
        append(&list1, data);
        break;
    case 2:
        printf("Add to list2 :");
        scanf("%d", &data);
        append(&list2, data);
        break;

    case 3: concat(&list1, list2);
        break;

    case 4: printf("list1 :");
        display(list1);
        break;
```

```
case 5: printf("List 2: ");
        display(list2);
        break;

case 6: printf("Exiting....\n");
        returno;

default:
        printf("invalid choice");
    }
}
returno;
}
```

Output(6a) →

```
Enter your choice: 1
Add to list 1: 8

Enter your choice: 1
Add to list 1: 7
Enter your choice: 1
Add to list: 9

Enter your choice: 2
Add to list2: 5
Enter your choice: 2
Add to list 2: 6
Enter your choice: 3
8 7 9 5 6
Enter your choice: 4
8 7 9
Enter your choice: 5
5 6
```

Code:

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

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

struct node*head = NULL;
void append(struct node* head  ,  int data)
{
   struct node* new_node = (struct node *)malloc(sizeof(struct node));
   new_node ->data = data;
   new_node->next = NULL;
   if(head == NULL)
   {
      head = new_node;
   }
   else
   {
      struct node*temp = head;
      while(temp->next != NULL)
      {
         temp = temp -> next;
      }
      temp -> next =new_node ;

   }


}

void sorting()
{
   struct node *ptr, *cpt;
   int temp;

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

```
    }

    for (ptr = head; ptr != NULL; ptr = ptr->next) {
        for (cpt = ptr->next; cpt != NULL; cpt = cpt->next) {
            if (ptr->data > cpt->data) {
                temp = ptr->data;
                ptr->data = cpt->data;
                cpt->data = temp;
            }
        }
    }
}


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



}

void reverse(struct node** head) {
    struct node* prev = NULL;
    struct node* curr = *head;
    struct node* next = NULL;
    while (curr != NULL) {
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
    *head = prev;
}
```

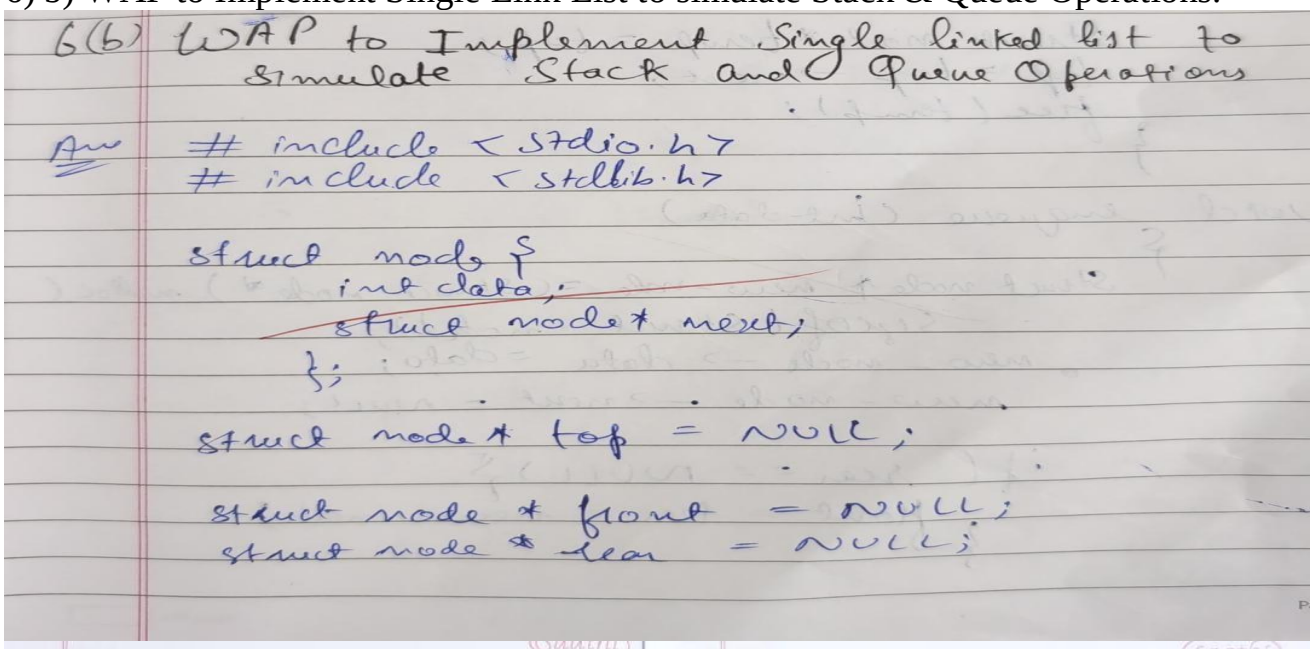
```
void display()
{
   struct node * temp;
   temp = head ;
   if (temp != NULL)
   {
      printf("list is empty");
   }
   else {
      while (temp != NULL)
      {
         printf("%d ->",temp->data);

      }
   }
}

int main() {
   struct node *list1 = NULL;
   struct node *list2 = NULL;
   int choice, data;

   while (1) {
      printf("1 for append to list1\n");
      printf("2 for append to list2\n");
      printf("3 for concatenation\n");
      printf("4 for display list1\n");
      printf("5 for display list2\n");
      printf("6 for reverse list1\n");
      printf("7 for reverse list2\n");
      printf("8 for exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch (choice) {
         case 1:
```

```
            printf("Add to list1: ");
            scanf("%d", &data);
            append(&list1, data);
            break;
        case 2:
            printf("Add to list2: ");
            scanf("%d", &data);
            append(&list2, data);
            break;
        case 3:
            concat(&list1, list2);
            break;
        case 4:
            printf("List1: ");
            display(list1);
            break;
        case 5:
            printf("List2: ");
            display(list2);
            break;
        case 6:
            printf("Reversing list1...\n");
            reverse(&list1);
            break;
        case 7:
            printf("Reversing list2...\n");
            reverse(&list2);
            break;
        case 8:
            printf("Exiting...\n");
            return 0;
        default:
            printf("Invalid choice\n");
            break;
    }
  }
}
```

```
1 for append to list1
2 for append to list2
3 for concatenation
4 for display list1
5 for display list2
6 for reverse list1
7 for reverse list2
8 for exit
Enter your choice: 1
Add to list1: 1
1 for append to list1
2 for append to list2
3 for concatenation
4 for display list1
5 for display list2
6 for reverse list1
7 for reverse list2
8 for exit
Enter your choice: 1
Add to list1: 2
1 for append to list1
2 for append to list2
3 for concatenation
4 for display list1
5 for display list2
6 for reverse list1
7 for reverse list2
8 for exit
Enter your choice: 2
Add to list2: 3
1 for append to list1
2 for append to list2
3 for concatenation
4 for display list1
5 for display list2
6 for reverse list1
7 for reverse list2
8 for exit
Enter your choice: 2
Add to list2: 34
1 for append to list1
2 for append to list2
3 for concatenation
4 for display list1
5 for display list2
6 for reverse list1
7 for reverse list2
8 for exit
```

```
Enter your choice: 7
Reversing list2...
1 for append to list1
2 for append to list2
3 for concatenation
4 for display list1
5 for display list2
6 for reverse list1
7 for reverse list2
8 for exit
Enter your choice: 4
List1: 2 -> 1 -> NULL
1 for append to list1
2 for append to list2
3 for concatenation
4 for display list1
5 for display list2
6 for reverse list1
7 for reverse list2
8 for exit
Enter your choice: 5
List2: 34 -> 3 -> NULL
1 for append to list1
2 for append to list2
3 for concatenation
4 for display list1
5 for display list2
6 for reverse list1
7 for reverse list2
8 for exit
Enter your choice: 3
1 for append to list1
2 for append to list2
3 for concatenation
4 for display list1
5 for display list2
6 for reverse list1
7 for reverse list2
8 for exit
Enter your choice: 4
List1: 2 -> 1 -> 34 -> 3 -> NULL
```

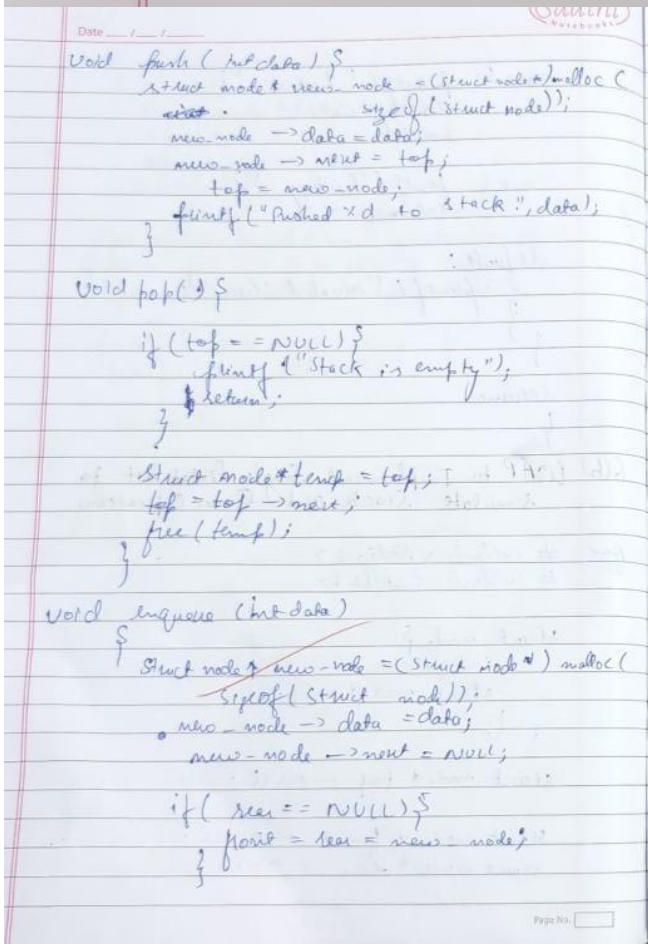
6) b) WAP to Implement Single Link List to simulate Stack & Queue Operations.



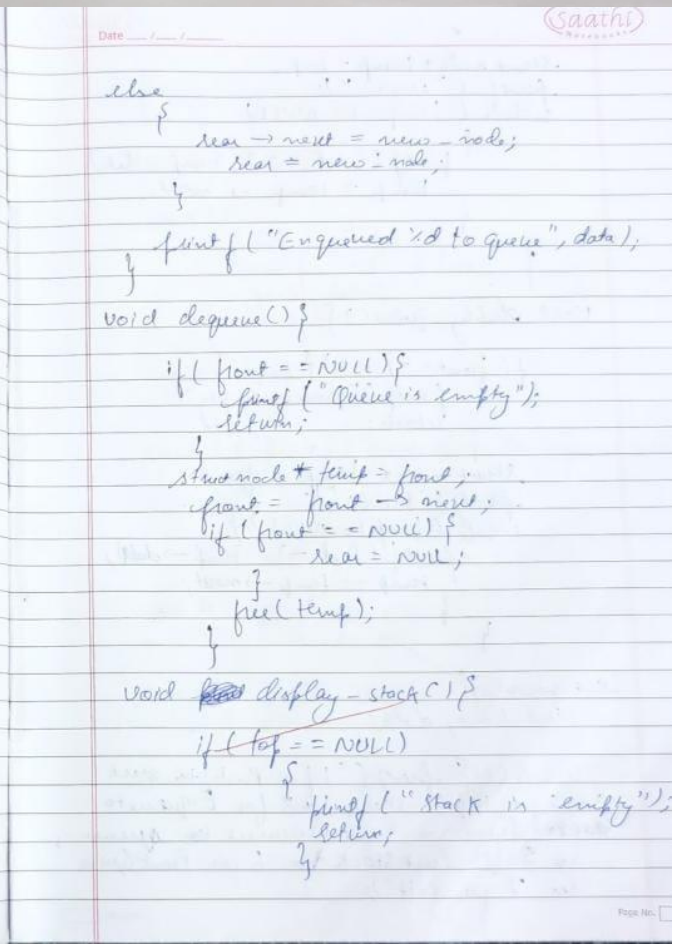
6(b) WAP to Implement Single linked list to simulate Stack and Queue Operations

Ans

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

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

struct node * top = NULL;

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



```
void push (int data) {
    struct node * new_node = (struct node *) malloc (
                sizeof (struct node));
    new_node -> data = data;
    new_node -> next = top;
    top = new_node;
    printf ("Pushed %d to stack", data);
}

void pop() {
    if (top == NULL) {
        printf ("Stack is empty");
        return;
    }
    struct node * temp = top;
    top = top -> next;
    free (temp);
}

void enqueue (int data)
{
    struct node * new_node = (struct node *) malloc (
        sizeof (struct node));
    new_node -> data = data;
    new_node -> next = NULL;
    if (rear == NULL) {
        front = rear = new_node;
    }
```



```
    else
    {
        rear -> next = new_node;
        rear = new_node;
    }
    printf ("Enqueued %d to queue", data);
}

void dequeue() {
    if (front == NULL) {
        printf ("Queue is empty");
        return;
    }
    struct node * temp = front;
    front = front -> next;
    if (front == NULL) {
        rear = NULL;
    }
    free (temp);
}

void display_stack() {
    if (top == NULL)
    {
        printf ("stack is empty");
        return;
    }
```

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

Output (6b) →

~~Printf ("enter your choice:");~~

enter your choice : 1
Enter data to push to stack : 1
Enter your choice : 1
enter data to push to stack : 2
Enter your choice : 1
enter data to push to stack : 3

Date ___/___/___

enter your choice : 3
enter data to enqueue into queue : 4
enter your choice : 3
enter data to enqueue into queue : 5
enter your choice : 3
enter data to enqueue into queue : 6
enter your choice : 2
enter your choice : 4
enter your choice : 5
1 2
enter your choice : 6
5 6
Enter your choice : 7
Exiting---

16/12/24

Code:

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

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

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

void push(int data) {
   struct node* new_node = (struct node*)malloc(sizeof(struct node));
   new_node->data = data;
   new_node->next = top;
   top = new_node;
   printf("Pushed %d to stack\n", data);
}

void pop() {
   if (top == NULL) {
      printf("Stack is empty\n");
      return;
   }
   struct node* temp = top;
   top = top->next;
   printf("Popped %d from stack\n", temp->data);
   free(temp);
}

void display_stack() {
   if (top == NULL) {
      printf("Stack is empty\n");
      return;
   }
   struct node* temp = top;
   printf("Stack: ");
   while (temp != NULL) {
```

```
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

void enqueue(int data) {
    struct node* new_node = (struct node*)malloc(sizeof(struct node));
    new_node->data = data;
    new_node->next = NULL;
    if (rear == NULL) {
        front = rear = new_node;
    } else {
        rear->next = new_node;
        rear = new_node;
    }
    printf("Enqueued %d to queue\n", data);
}

void dequeue() {
    if (front == NULL) {
        printf("Queue is empty\n");
        return;
    }
    struct node* temp = front;
    front = front->next;
    if (front == NULL) {
        rear = NULL;
    }
    printf("Dequeued %d from queue\n", temp->data);
    free(temp);
}

void display_queue() {
    if (front == NULL) {
        printf("Queue is empty\n");
        return;
    }
    struct node* temp = front;
    printf("Queue: ");
```

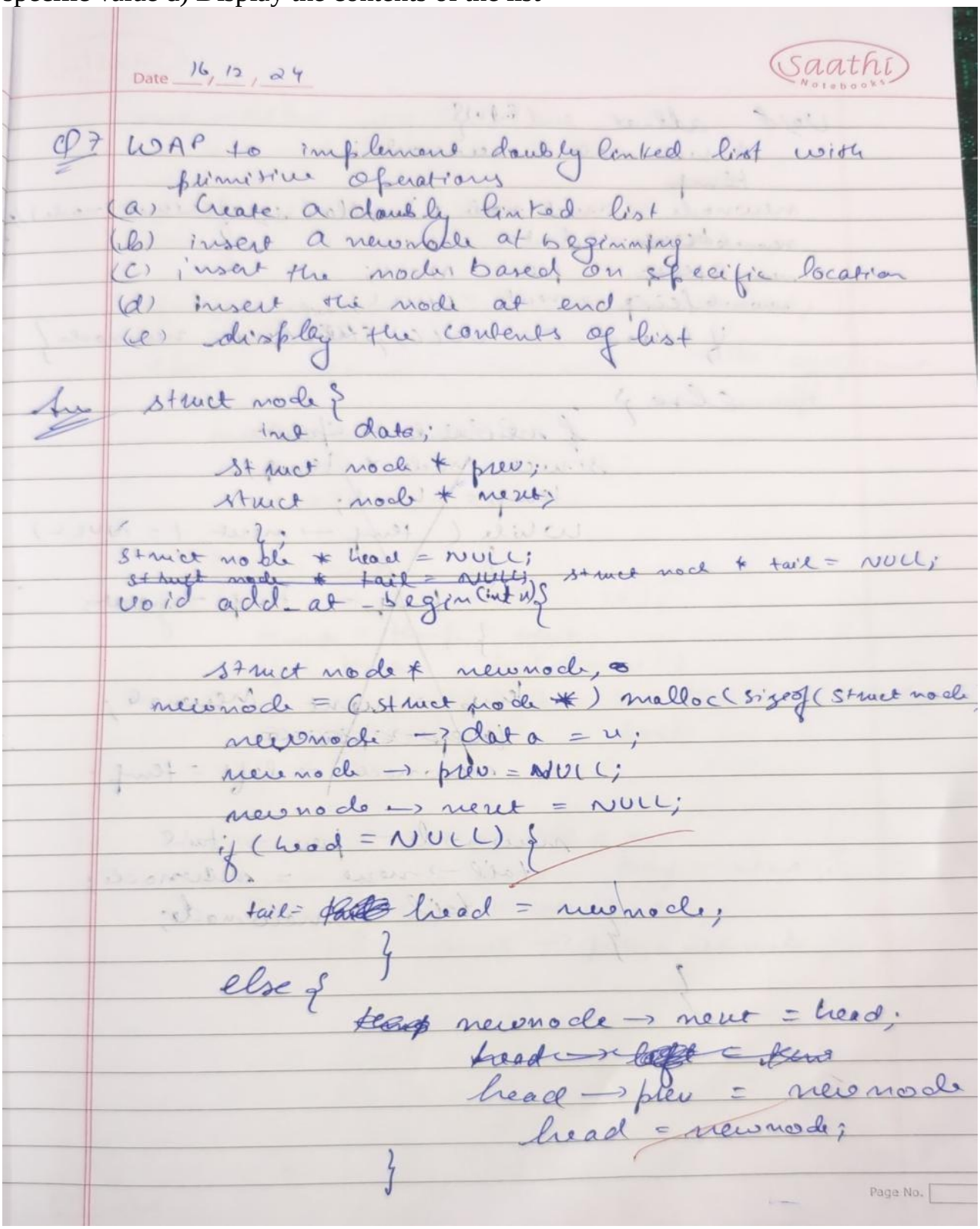
```
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
    int choice, data;
    while (1) {
        printf("\n1. Push to Stack\n");
        printf("2. Pop from Stack\n");
        printf("3. Display Stack\n");
        printf("4. Enqueue to Queue\n");
        printf("5. Dequeue from Queue\n");
        printf("6. Display Queue\n");
        printf("7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to push to stack: ");
                scanf("%d", &data);
                push(data);
                break;
            case 2:
                pop();
                break;
            case 3:
                display_stack();
                break;
            case 4:
                printf("Enter data to enqueue to queue: ");
                scanf("%d", &data);
                enqueue(data);
                break;
            case 5:
                dequeue();
                break;
```

```
            case 6:
                display_queue();
                break;
            case 7:
                printf("Exiting...\n");
                return 0;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }
    return 0;
}
```

```
1. Push to Stack
2. Pop from Stack
3. Display Stack
4. Enqueue to Queue
5. Dequeue from Queue
6. Display Queue
7. Exit
Enter your choice: 1
Enter data to push to stack: 1
Pushed 1 to stack

1. Push to Stack
2. Pop from Stack
3. Display Stack
4. Enqueue to Queue
5. Dequeue from Queue
6. Display Queue
7. Exit
Enter your choice: 1
Enter data to push to stack: 2
Pushed 2 to stack

1. Push to Stack
2. Pop from Stack
3. Display Stack
4. Enqueue to Queue
5. Dequeue from Queue
6. Display Queue
7. Exit
Enter your choice: 1
Enter data to push to stack: 3
Pushed 3 to stack

1. Push to Stack
2. Pop from Stack
3. Display Stack
4. Enqueue to Queue
5. Dequeue from Queue
6. Display Queue
7. Exit
Enter your choice: 2
Popped 3 from stack

1. Push to Stack
2. Pop from Stack
3. Display Stack
4. Enqueue to Queue
5. Dequeue from Queue
6. Display Queue
7. Exit
Enter your choice: 3
Stack: 2 -> 1 -> NULL
```

```
1. Push to Stack
2. Pop from Stack
3. Display Stack
4. Enqueue to Queue
5. Dequeue from Queue
6. Display Queue
7. Exit
Enter your choice: 4
Enter data to enqueue to queue: 7
Enqueued 7 to queue

1. Push to Stack
2. Pop from Stack
3. Display Stack
4. Enqueue to Queue
5. Dequeue from Queue
6. Display Queue
7. Exit
Enter your choice: 4
Enter data to enqueue to queue: 9
Enqueued 9 to queue

1. Push to Stack
2. Pop from Stack
3. Display Stack
4. Enqueue to Queue
5. Dequeue from Queue
6. Display Queue
7. Exit
Enter your choice: 5
Dequeued 7 from queue

1. Push to Stack
2. Pop from Stack
3. Display Stack
4. Enqueue to Queue
5. Dequeue from Queue
6. Display Queue
7. Exit
Enter your choice: 6
Queue: 9 -> NULL

1. Push to Stack
2. Pop from Stack
3. Display Stack
4. Enqueue to Queue
5. Dequeue from Queue
6. Display Queue
7. Exit
Enter your choice:
```

7) WAP to Implement doubly link list with primitive operations a) Create a doubly linked list. b) Insert a new node to the left of the node. c) Delete the node based on a specific value d) Display the contents of the list



Date 16, 12, 24

Saathi Notebooks

Q7 WAP to implement doubly linked list with primitive operations
(a) Create a doubly linked list
(b) insert a new node at beginning
(c) insert the node based on specific location
(d) insert the node at end
(e) display the contents of list

```
struct node {
    int data;
    struct node * prev;
    struct node * next;
};
struct node * head = NULL;
struct node * tail = NULL;
void add_at_begin(int u){

    struct node * newnode;
newnode = (struct node *) malloc(sizeof(struct node
    newnode -> data = u;
    newnode -> prev = NULL;
    newnode -> next = NULL;
    if (head = NULL) {
    tail = head = newnode;
    }
else {
    newnode -> next = head;
    head -> prev = newnode
    head = newnode;
}
```

Page No.

```
void add_at_end(int u){
    struct node *newnode;
    newnode = (struct node *)malloc(sizeof(struct node));
    newnode -> data = u;
    newnode -> prev = NULL;
    newnode -> next = NULL;
    if (head == NULL) head = newnode;
    else {
        struct node *temp;
        temp = head;
        while (temp -> next != NULL)
            temp = temp -> next;
        temp -> next = newnode;
        newnode -> left = temp;
        newnode -> prev = tail
        tail -> next = newnode;
        tail = newnode;
    }
}
```

```
void insertpos(int u){
    struct node *newnode, *temp;
    int pos;
    printf("enter the position:");
    scanf("%d", &pos);
    if (pos == 1) { void insert_at_beg(u); }
    else {
        newnode = (struct node *) malloc(sizeof(struct node));
        newnode -> data = u;
        newnode -> next = NULL;
        newnode -> prev = NULL;
        temp = head;
        for (int i = 1; i < pos - 1; i++){
            temp = temp -> next;
            if (temp == tail -> next) {
                printf("There are less than %d nodes", pos);
                return;
            }
        }
        newnode -> prev = temp;
        newnode -> next = temp -> next;
        temp -> next = newnode;
        newnode -> next -> prev = newnode;
    }
}
```

```
void printlist(){
    struct node *temp;
    temp = head;
    if (temp == NULL) { printf("list is empty"); }
    else {
        while (temp != NULL) {
            printf("%d <-> ", temp -> data);
            temp = temp -> next;
        }
    }
}

int main(){
    int choice, data, pos;
    while (1){
        printf("1 for insert at beginning \n
        2 for insert at specific position \n
        3 for insert at end \n 4 for display list
        \n 5 for exit");
        printf("enter your choice:");
        scanf("%d", &choice);
        switch(choice){
        case 1: printf("enter data to insert at
                beginning:");
            scanf("%d", &data);
            add_at_beg(data);
            break;
        case 2: printf("enter data to insert:");
            scanf("%d", &data);
            insertpos(u);
            break;
        case 3: printf("enter data to insert at
                end:");
            scanf("%d", &data);
            add_at_end(data);
            break;
        case 4: display();
            break;
        case 5: printf("Exiting.....\n");
            exit(0);
        default: printf("Invalid Choice. Pls try again\n");
        }
    }
    return 0;
}
```

Saathi Notebooks

Date __/__/____

Output:→

Doubly linked list operations
1 Add at beginning
2 Add at end
3 Insert at position
4 Display list
5 Exit
Enter your choice: 1
Enter data to insert at beginning: 22
Enter your choice: 1
Enter data to " " " : 223
Enter your choice: 2
Enter data to insert at end: 67
Enter your choice: 2
Enter data to add at end: 55
Enter your choice: 3
Enter data to insert at position: 33
Enter the position: 1
Enter your choice: 4
33 ↔ 223 ↔ 22 ↔ 67 → 55 ↔ NULL
Enter your choice: 5
Exiting....

Geeta
16/12/24

Code:

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

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

struct node *head = NULL;
struct node *tail = NULL;

void add_at_begin(int x) {
   struct node *newnode;
   newnode = (struct node *)malloc(sizeof(struct node));
   newnode->data = x;
   newnode->next = NULL;
   newnode->prev = NULL;
   if (head == NULL) {
      head = tail = newnode;
   } else {
      newnode->next = head;
      head->prev = newnode;
      head = newnode;
   }
}

void add_at_end(int x) {
   struct node *newnode;
   newnode = (struct node *)malloc(sizeof(struct node));
   newnode->data = x;
   newnode->next = NULL;
   newnode->prev = NULL;
   if (head == NULL) {
      head = tail = newnode;
   } else {
      newnode->prev = tail;
```

```
        tail->next = newnode;
        tail = newnode;
    }
}

void insertpos(int x) {
    struct node *newnode, *temp;
    int pos;
    printf("Enter the position: ");
    scanf("%d", &pos);

    if (pos == 1) {
        add_at_begin(x);
    } else {
        newnode = (struct node *)malloc(sizeof(struct node));
        newnode->data = x;
        newnode->next = NULL;
        newnode->prev = NULL;
        temp = head;

        for (int i = 1; i < pos - 1; i++) {
            temp = temp->next;
            if (temp == NULL) {
                printf("There are less than %d nodes.\n", pos);
                free(newnode);
                return;
            }
        }

        newnode->prev = temp;
        newnode->next = temp->next;
        if (temp->next != NULL) {
            temp->next->prev = newnode;
        }
        temp->next = newnode;

        if (newnode->next == NULL) {
            tail = newnode;
        }
    }
```

```
}

void printlist() {
    struct node *temp;
    temp = head;
    if (temp == NULL) {
        printf("The list is empty.\n");
    } else {
        while (temp != NULL) {
            printf("%d <-> ", temp->data);
            temp = temp->next;
        }
        printf("NULL\n");
    }
}

int main() {
    int choice, value;

    while (1) {
        printf("\nDoubly Linked List Menu:\n");
        printf("1. Add at the beginning\n");
        printf("2. Add at the end\n");
        printf("3. Insert at a position\n");
        printf("4. Print the list\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to add at the beginning: ");
                scanf("%d", &value);
                add_at_begin(value);
                break;
            case 2:
                printf("Enter value to add at the end: ");
                scanf("%d", &value);
                add_at_end(value);
                break;
```

```
            case 3:
                printf("Enter value to insert at a position: ");
                scanf("%d", &value);
                insertpos(value);
                break;
            case 4:
                printlist();
                break;
            case 5:
                printf("Exiting...\n");
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

```
Doubly Linked List Menu:
1. Add at the beginning
2. Add at the end
3. Insert at a position
4. Print the list
5. Exit
Enter your choice: 1
Enter value to add at the beginning: 22

Doubly Linked List Menu:
1. Add at the beginning
2. Add at the end
3. Insert at a position
4. Print the list
5. Exit
Enter your choice: 1
Enter value to add at the beginning: 45

Doubly Linked List Menu:
1. Add at the beginning
2. Add at the end
3. Insert at a position
4. Print the list
5. Exit
Enter your choice: 2
Enter value to add at the end: 78

Doubly Linked List Menu:
1. Add at the beginning
2. Add at the end
3. Insert at a position
4. Print the list
5. Exit
Enter your choice: 3
Enter value to insert at a position: 1
Enter the position: 1

Doubly Linked List Menu:
1. Add at the beginning
2. Add at the end
3. Insert at a position
4. Print the list
5. Exit
Enter your choice: 4
1 <-> 45 <-> 22 <-> 78 <-> NULL
```

8) Write a program a) To construct a binary Search tree. b) To traverse the tree using all the methods i.e., in order, preorder and post order c) To display the elements in the tree

Saathi Notebooks

Date 23 / 12 / 2024

Q8 Write a Program
(a) To construct a binary search tree
(b). To traverse the tree using inorder, preorder + postorder, display all traversal

Ans

```
struct Node { int key;
              struct Node * left, * right;
            };

struct Node * newNode( int key) {
    struct node * temp = (struct Node *) malloc( sizeof(
                         struct node));
    temp → key = key
    temp → left = NULL;
    temp → right = NULL;
    return temp;
}

void inOrder traversal( struct node * root) {
    if ( root != NULL) {
        inOrdertraversal (root → left);
        printf ("%d", root → key);
        inOrdertraversal (root → right);
    }
}

void preOrder traversal ( struct node * root ) {
    if ( root != NULL ) {
        printf ( "%d" , root → key);
        preOrdertraversal ( root → left );
        preOrdertraversal ( root → right);
    }
}
```

Page No.

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

int main(){
    struct Node* root = NULL;
int elements[] = {50, 30, 20, 40, 70, 60, 80};
int n = sizeof(elements)/sizeof(elements[0]);
for(int i=0; i<n; i++){
        root = insert(root, elements[i]);
    }
printf("In Order Traversal :");
inOrdertraversal(root);
preOrdertraversal(root);
printf("Pre-order traversal:");
postOrdertraversal(root);
printf("Post-order traversal:");
printf("\n");
return 0;
}
```

Output:→ In-Order Traversal: 20 30 40 50 60 70 80

Pre-Order Traversal: 50 30 20 40 70 60 80

Post-Order Traversal: 20 40 30 60 80 70 50

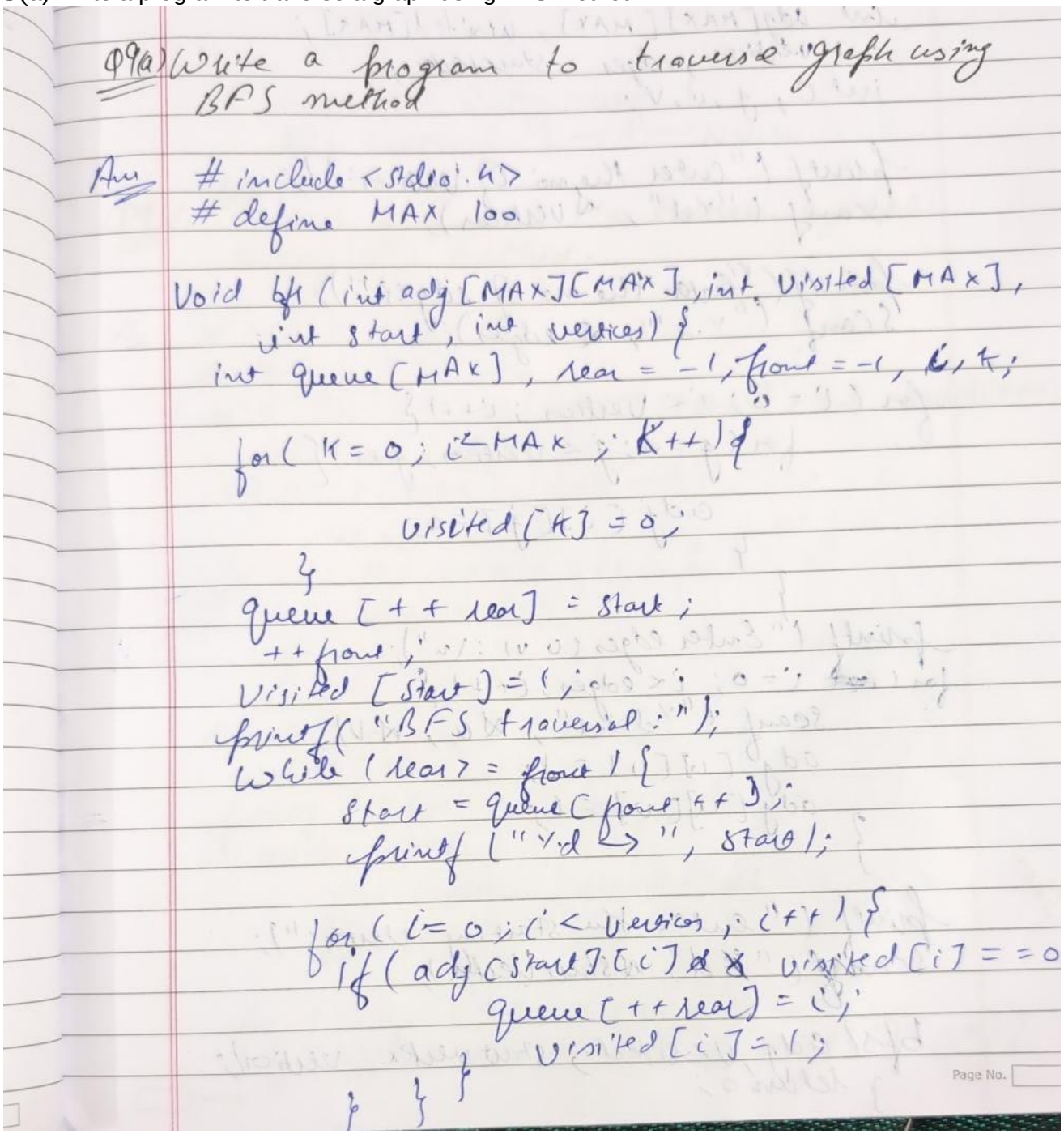
Code:

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

struct Node {
    int key;
    struct Node *left, *right;
};

struct Node* newNode(int key) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->key = key;
    node->left = node->right = NULL;
    return node;
}

struct Node* insert(struct Node* node, int key) {
    if (node == NULL) return newNode(key);

    if (key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);

    return node;
}

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

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

```
}

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

int main() {
    struct Node* root = NULL;

    int elements[] = {50, 30, 20, 40, 70, 60, 80};
    int n = sizeof(elements) / sizeof(elements[0]);

    for (int i = 0; i < n; i++) {
        root = insert(root, elements[i]);
    }

    printf("In-order Traversal: ");
    inOrderTraversal(root);
    printf("\n");

    printf("Pre-order Traversal: ");
    preOrderTraversal(root);
    printf("\n");

    printf("Post-order Traversal: ");
    postOrderTraversal(root);
    printf("\n");

    return 0;
}
```

```
In-order Traversal: 20 30 40 50 60 70 80
Pre-order Traversal: 50 30 20 40 70 60 80
Post-order Traversal: 20 40 30 60 80 70 50

Process returned 0 (0x0)   execution time : 0.005 s
Press any key to continue.
```

9(a) Write a program to traverse a graph using BFS method.



Q9(a) Write a program to traverse graph using BFS method

Ans

```
#include <stdio.h>
#define MAX 100

Void bfs (int adj[MAX][MAX], int visited[MAX],
    int start, int vertices) {
int queue[MAX], rear = -1, front = -1, i, k;

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

        visited[k] = 0;
    }
queue[++rear] = start;
++front;
visited[start] = 1;
printf("BFS traversal :");
while (rear >= front) {
    start = queue[front++];
    printf("%d -> ", start);

for( i= 0; i < vertices ; i++) {
if( adj[start][i] && visited[i] == 0
        queue[++rear] = i;
        visited[i] = 1;
} } }
```

```
    printf("Null \n");
}
int main(){
    int adj[MAX][MAX], visited[MAX];
    int vertices, edges, startVertex;
    int i, j, u, v;
    printf("Enter the no of vertices:");
    scanf("%d", &vertices);
    printf("Enter the no of edges:");
    scanf("%d", &edges);
    for(i=0; i<vertices; i++){
        for(j=0; j<vertices; j++){
            adj[i][j]=0;
        }
    }
    printf("Enter edges (u v):\n");
    for(i=0; i<edges; i++){
        scanf("%d %d", &u, &v);
        adj[u][v]=1;
        adj[v][u]=1;
    }
    printf("enter the starting vertex:");
    scanf("%d", &startVertex);
    bfs(adj, visited, startVertex, vertices);
    return 0;
```

Output :→ Enter the no of vertices : 7
Enter the no of edges : 4
Enter the edges (u v):
1 2
2 5
3 6
4 5
Enter the starting vertex : 1
BFS Traversal : 1 → 2 → NULL

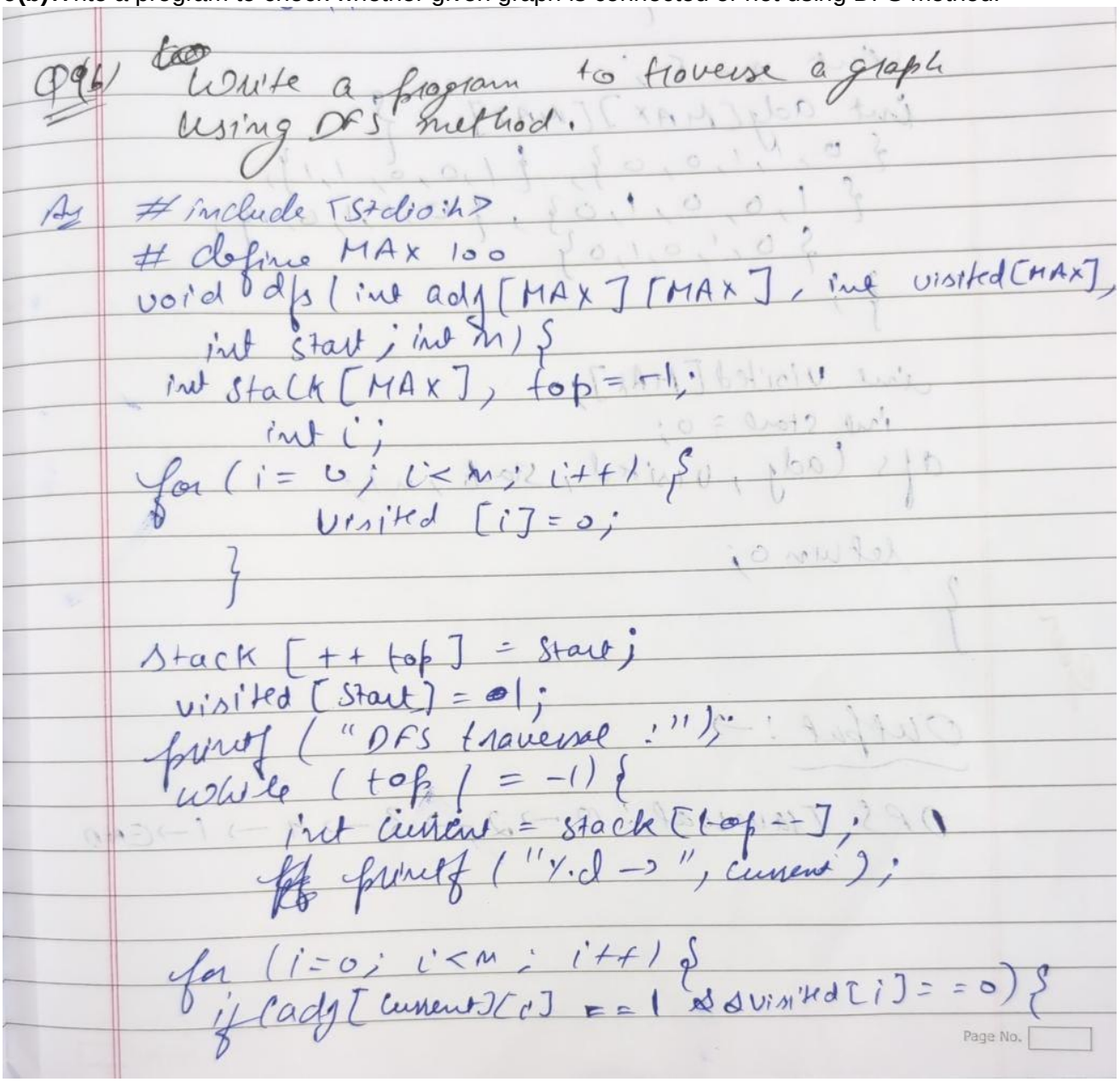
Code:

```
#include <stdio.h>
#define MAX 8

void bfs(int adj[MAX][MAX], int visited[MAX], int start) {
    int queue[MAX], rear = -1, front = -1, i;

    for (i = 0; i < MAX; i++) {
        visited[i] = 0;
    }

    queue[++rear] = start;
    ++front;
    visited[start] = 1;


    while (rear >= front) {
        start = queue[front++];
        printf("%d -> ", start);

        for (i = 0; i < MAX; i++) {
            if (adj[start][i] && visited[i] == 0) {
                queue[++rear] = i;
                visited[i] = 1;
            }
        }
    }
    printf("NULL\n");
}



int main() {
    int visited[MAX] = {0};
    int adj[MAX][MAX] = {
        {0, 1, 1, 0, 0, 0, 0, 0},
        {1, 0, 0, 1, 0, 0, 0, 0},
        {1, 0, 0, 1, 0, 0, 0, 0},
        {0, 1, 1, 0, 1, 1, 0, 0},
        {0, 0, 0, 1, 0, 0, 1, 1},
        {0, 0, 0, 1, 0, 0, 1, 1},
        {0, 0, 0, 0, 1, 1, 0, 0},
        {0, 0, 0, 0, 1, 1, 0, 0},
    };

    printf("BFS Traversal\n");
    bfs(adj, visited, 0);

    for (int i = 0; i < MAX; i++) {
        visited[i] = 0;
    }
```

```
    return 0;
}
```

```
BFS Traversal
0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> NULL

Process returned 0 (0x0)   execution time : 0.007 s
Press any key to continue.
```

9(b) Write a program to check whether given graph is connected or not using DFS method.



Q9(b) Write a program to traverse a graph using DFS method.

Ans

```
#include<stdio.h>
#define MAX 100
void dfs(int adj[MAX][MAX], int visited[MAX],
   int start; int n) {
 int stack[MAX], top=-1;
     int i;
for(i=0; i<n; i++) {
        visited[i]=0;
    }

stack[++top] = start;
 visited[start]=1;
printf("DFS traversal :");
while(top!=-1){
     int current = stack[top--];
     printf("%d -> ", current);

for(i=0; i<n; i++) {
 if(adj[current][i]==1 && visited[i]==0){
```

```
            stack[++top] = i;
            visited[i] = 1;
        }
    }
    printf("END\n");
}

int main() {
    int n = 5;
    int adj[MAX][MAX] = {
        {0,1,1,0,0}, {1,0,0,1,1},
        {1,0,0,1,0}, {0,1,1,0,1},
        {0,1,0,1,0}
    };
    int visited[MAX];
    int start = 0;
    dfs(adj, visited, start, n);

    return 0;
}
```

Seen

Output :→

DFS Traversal: 0 → 2 → 3 → 4 → 1 → END

Code:

```
#include <stdio.h>

#define MAX 100

void dfs(int adj[MAX][MAX], int visited[MAX], int start, int n) {
    int stack[MAX], top = -1;
    int i;

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

    stack[++top] = start;
    visited[start] = 1;

    printf("DFS Traversal: ");

    while (top != -1) {
        int current = stack[top--];
        printf("%d -> ", current);

        for (i = 0; i < n; i++) {
            if (adj[current][i] == 1 && visited[i] == 0) {
                stack[++top] = i;
                visited[i] = 1;
            }
        }
    }

    printf("NULL\n");
}

int main() {
    int n = 5;
    int adj[MAX][MAX] = {
        {0, 1, 1, 0, 0},
        {1, 0, 0, 1, 1},
        {1, 0, 0, 1, 0},
        {0, 1, 1, 0, 1},
```

```
        {0, 1, 0, 1, 0}
    };
    int visited[MAX];

    int start = 0;
    dfs(adj, visited, start, n);

    return 0;
}
```

```
DFS Traversal: 0 -> 2 -> 3 -> 4 -> 1 -> NULL

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