VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Data Structures using C Lab

(23CS3PCDST)

Submitted by

SIDDHARTH ARYA (1BM23CS328)

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
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 **SIDDHARTH ARYA (1BM23CS328),** 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. 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.

Dr. Prasad G R Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

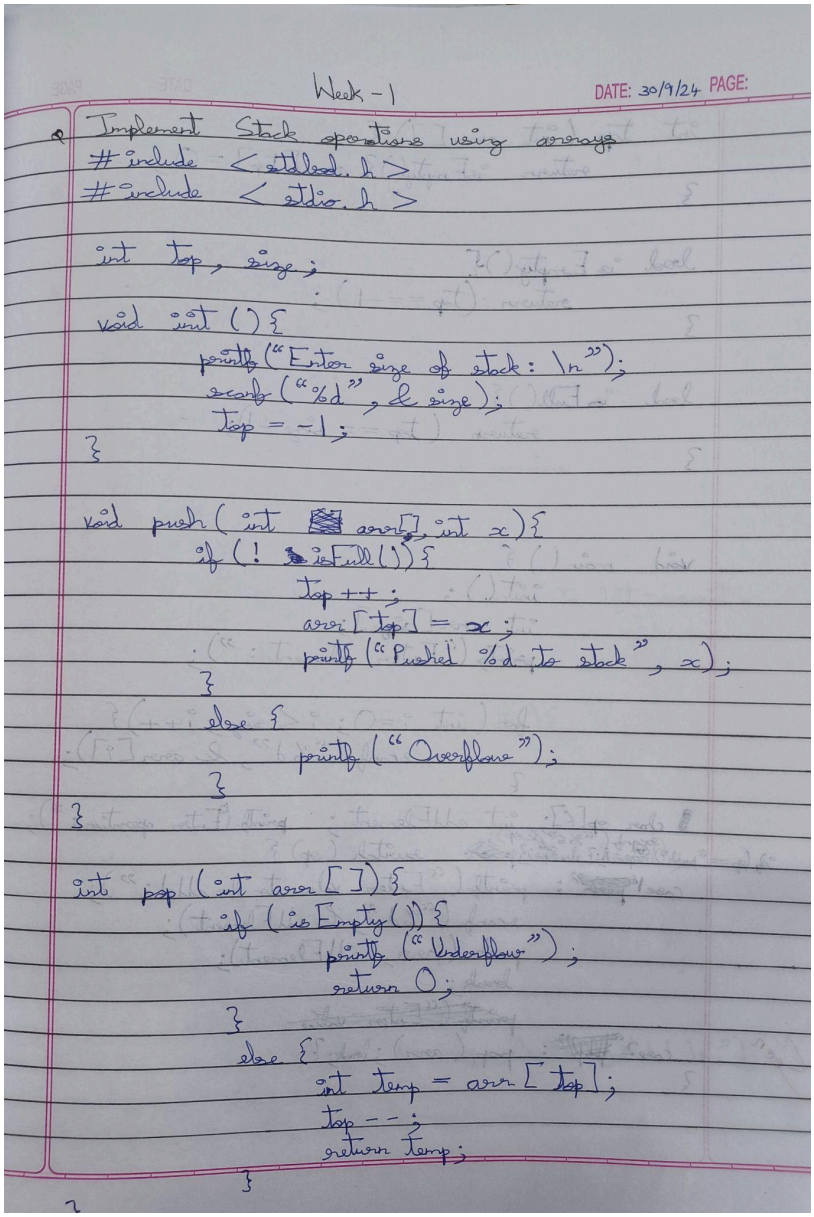
Index

Github Link:
https://github.com/Sid-CS328/1BM23CS328-DataStructuresWithC

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

Screenshot of Observation:



Week - 1 DATE: 30/9/24 PAGE:

Implement Stack operations using arrays

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

int top, size;

void init (){
    printf("Enter size of stack: \n");
    scanf("%d", &size);
    top = -1;
}

void push(int arr[], int x){
    if(! isFull()){
        top++;
        arr[top] = x;
        printf("Pushed %d to stack", x);
    }
    else {
        printf("Overflow");
    }
}

int pop(int arr[]){
    if(isEmpty()){
        printf("Underflow");
        return 0;
    }
    else {
        int temp = arr[top];
        top--;
        return temp;
    }
}
```

Code:

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

#define MAX 5


struct Stack {
   int arr[MAX];
   int top;
};


void initializeStack(struct Stack *s) {
   s->top = -1;
}


int isFull(struct Stack *s) {
   return s->top == MAX - 1;
}


int isEmpty(struct Stack *s) {
   return s->top == -1;
}


void push(struct Stack *s, int value) {
   if (isFull(s)) {
      printf("Stack Overflow! Cannot push %d.\n", value);
   }
   else {
      s->top++;
      s->arr[s->top] = value;
      printf("%d pushed to stack.\n", value);
   }
}


int pop(struct Stack *s) {
   if (isEmpty(s)) {
      printf("Stack Underflow! Cannot pop.\n");
      return -1;
   }
   else {
      int poppedValue = s->arr[s->top];
      s->top--;
      return poppedValue;
   }
```

```
}


void display(struct Stack *s) {
   if (isEmpty(s)) {
      printf("Stack is empty. Nothing to display.\n");
   }
   else {
      printf("Stack contents: ");
      for (int i = 0; i <= s->top; i++) {
         printf("%d ", s->arr[i]);
      }
      printf("\n");
   }
}

int main() {
   struct Stack stack;
   initializeStack(&stack);
   int choice, value;

   while (1) {
      printf("\nStack Operations Menu:\n");
      printf("1. Push\n");
      printf("2. Pop\n");
      printf("3. Display\n");
      printf("4. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch (choice) {
         case 1:

            printf("Enter value to push: ");
            scanf("%d", &value);
            push(&stack, value);
            break;
         case 2:

            value = pop(&stack);
            if (value != -1) {
               printf("Popped value: %d\n", value);
            }
            break;
         case 3:

            display(&stack);
            break;
         case 4:
```

```
            printf("Exiting program.\n");
            exit(0);
            break;
        default:
            printf("Invalid choice! Please try again.\n");
    }
  }

  return 0;
}
```

Output:

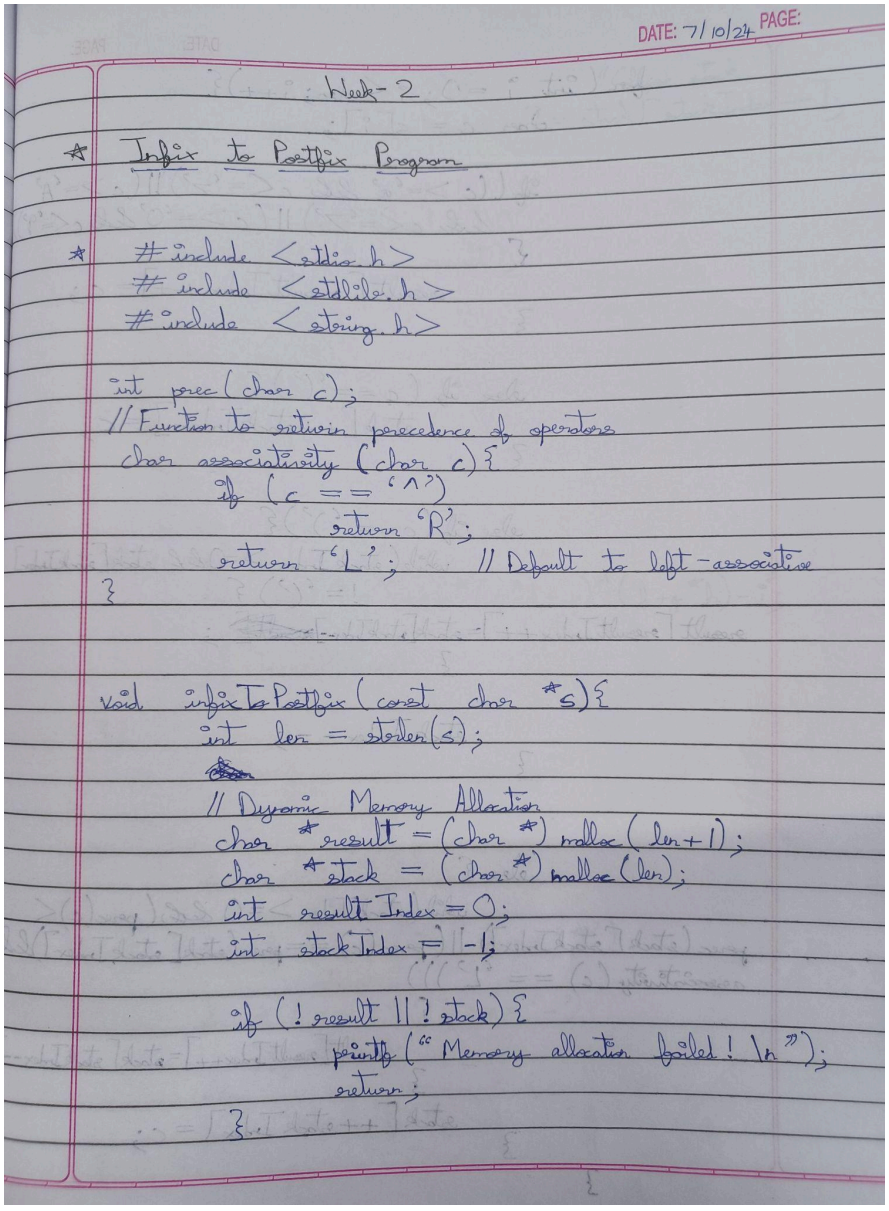
```
Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 12
12 pushed to stack.

Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 15
15 pushed to stack.

Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack contents: 12 15
```

```
Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack contents: 12

Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 4
Exiting program.
```

Lab Program 2

Write a program 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).

Screenshot of Observation:

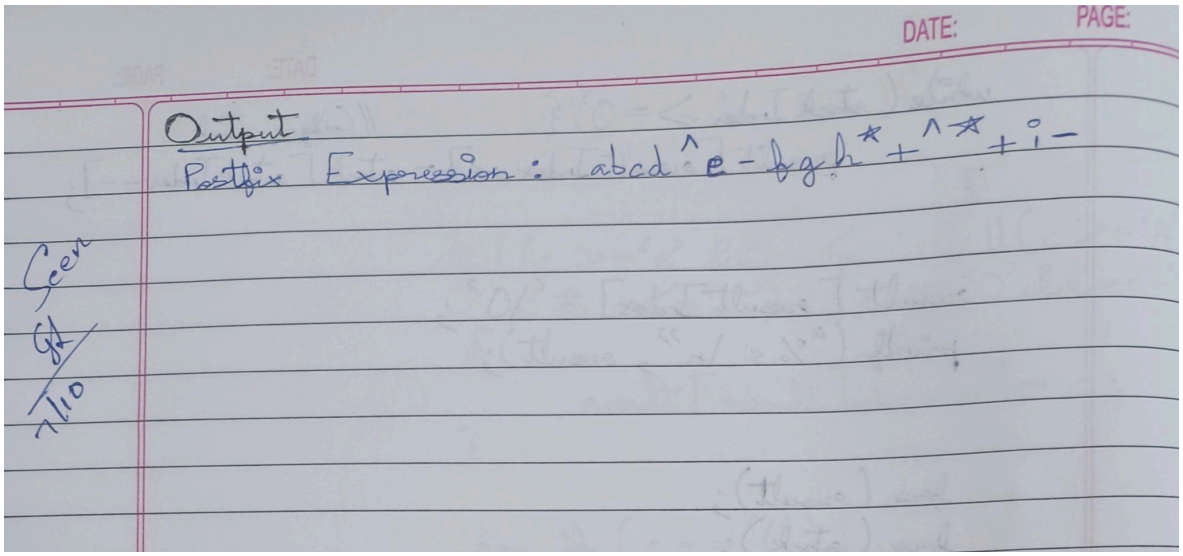


DATE: 7/10/24 PAGE:

Week-2

★ Infix to Postfix Program

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

int prec(char c);
// Function to return precedence of operators
char associativity(char c){
    if (c == '^')
        return 'R';
    return 'L';     // Default to left-associative
}

void infixToPostfix(const char *s){
    int len = strlen(s);

    // Dynamic Memory Allocation
    char *result = (char *) malloc(len+1);
    char *stack = (char *) malloc(len);
    int resultIndex = 0;
    int stackIndex = -1;

    if (!result || !stack){
        printf("Memory allocation failed!\n");
        return;
    }
```



Code:

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

#define MAX 100


struct Stack {
    char arr[MAX];
    int top;
};


void initStack(struct Stack* s) {
    s->top = -1;
}


int isEmpty(struct Stack* s) {
    return s->top == -1;
}


int isFull(struct Stack* s) {
    return s->top == MAX - 1;
}


void push(struct Stack* s, char c) {
    if (isFull(s)) {
        printf("Stack overflow\n");
        exit(1);
```

```
    }
    s->arr[++(s->top)] = c;
}


char pop(struct Stack* s) {
    if (isEmpty(s)) {
        printf("Stack underflow\n");
        exit(1);
    }
    return s->arr[(s->top)--];
}


char peek(struct Stack* s) {
    if (isEmpty(s)) {
        return -1;
    }
    return s->arr[s->top];
}


int precedence(char c) {
    if (c == '+' || c == '-') {
        return 1;
    } else if (c == '*' || c == '/') {
        return 2;
    }
    return 0;
}


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


void infixToPostfix(char* infix, char* postfix) {
    struct Stack s;
    initStack(&s);
    int i = 0, j = 0;
    char currentChar;

    while ((currentChar = infix[i++]) != '\0') {

        if (isalnum(currentChar)) {
            postfix[j++] = currentChar;
        }
```

```
        else if (currentChar == '(') {
            push(&s, currentChar);
        }

        else if (currentChar == ')') {
            while (!isEmpty(&s) && peek(&s) != '(') {
                postfix[j++] = pop(&s);
            }
            pop(&s);
        }

        else if (isOperator(currentChar)) {
            while (!isEmpty(&s) && precedence(peek(&s)) >= precedence(currentChar)) {
                postfix[j++] = pop(&s);
            }
            push(&s, currentChar);
        }
    }


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

    postfix[j] = '\0';
}


int main() {
    char infix[MAX], postfix[MAX];

    printf("Enter a valid infix expression: ");
    scanf("%s", infix);

    infixToPostfix(infix, postfix);

    printf("Postfix expression: %s\n", postfix);

    return 0;
}
```

Output:

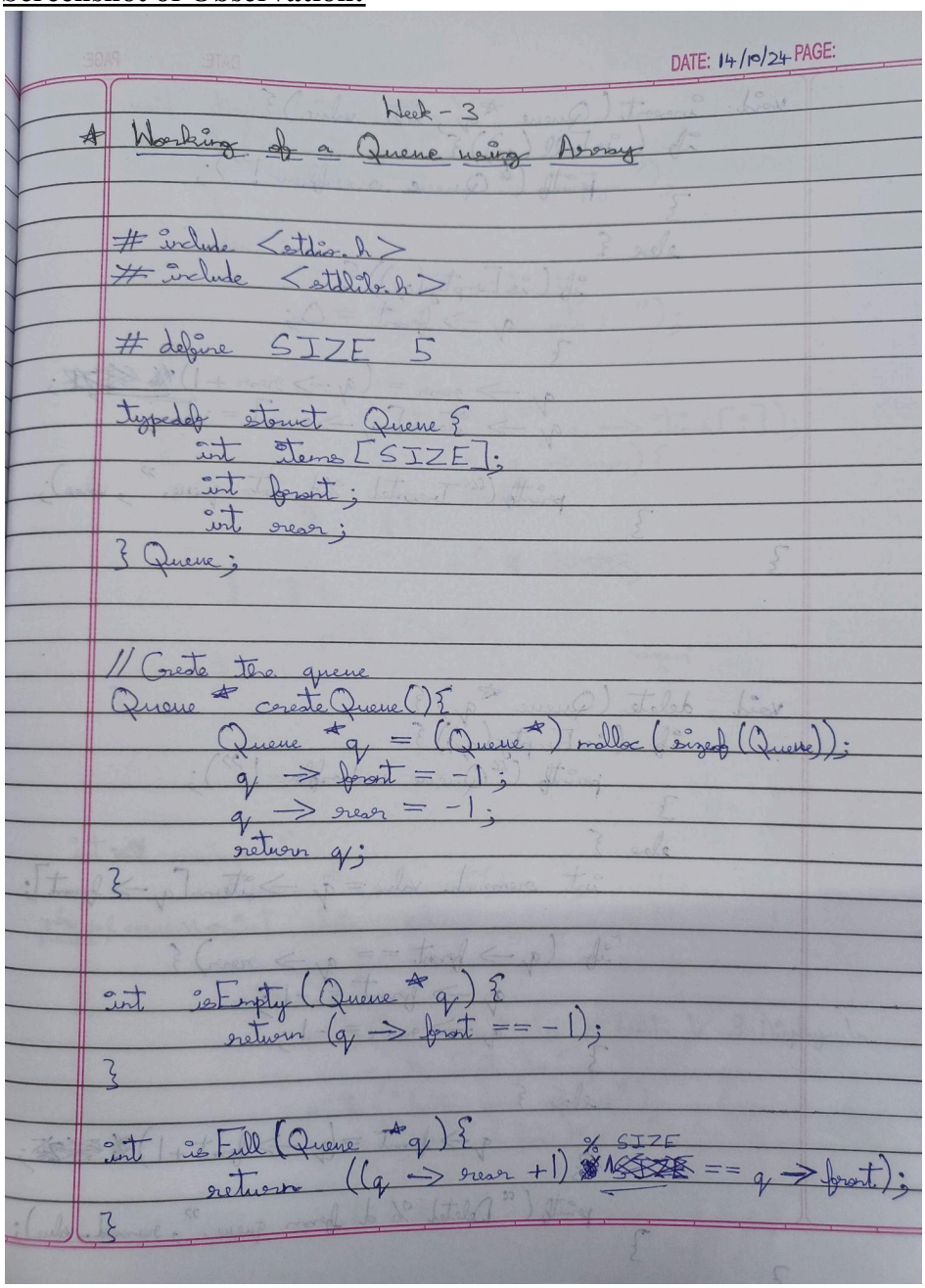
```
Enter a valid infix expression: a+b*(c+d-e)/(f+g*h)-i
Postfix expression: abcd+e-*fgh*+/+i-
```

Lab Program 3

3a) Write a program 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

3b) Write a program 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.

Screenshot of Observation:

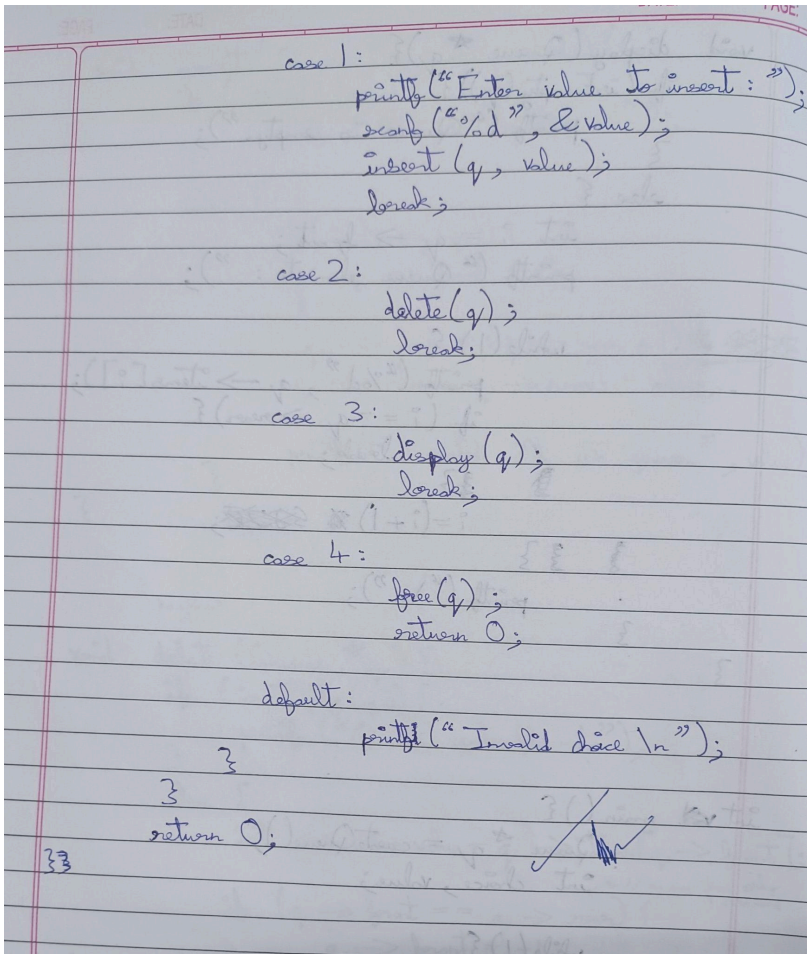


DATE: 14/10/24 PAGE:

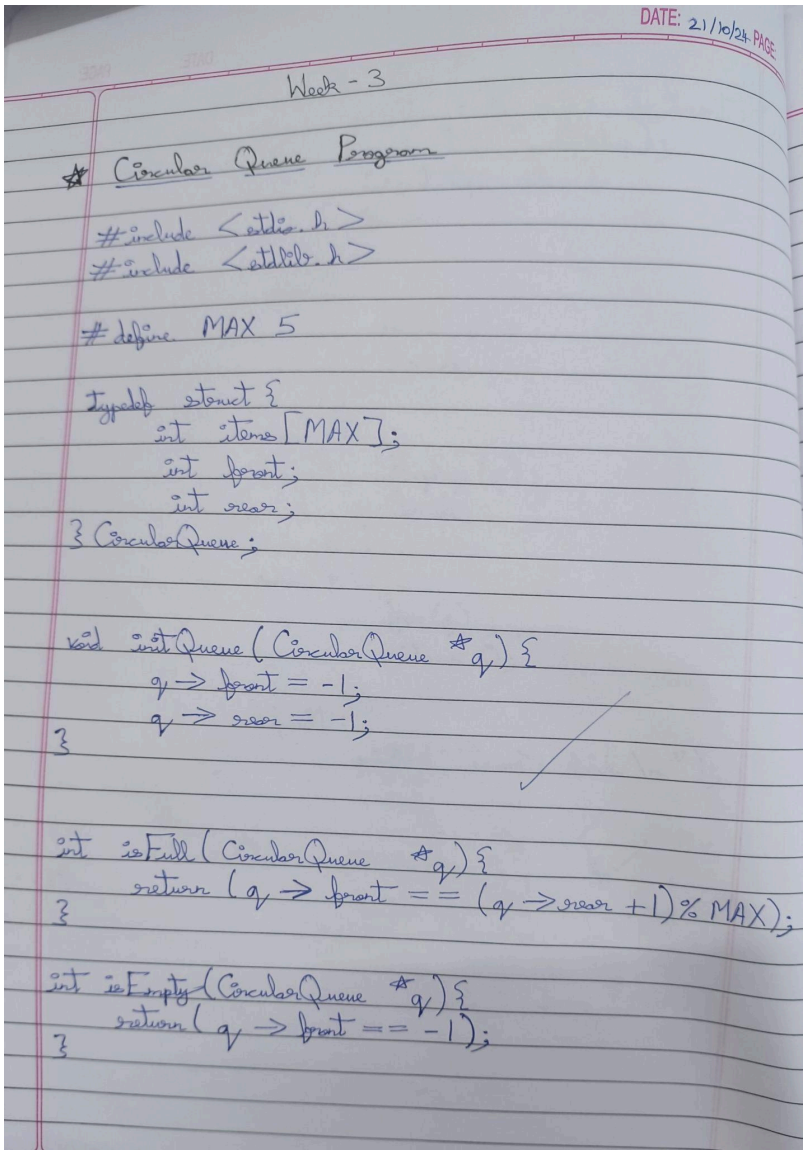
Week - 3

* Working of a Queue using Array

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

#define SIZE 5

typedef struct Queue {
    int items[SIZE];
    int front;
    int rear;
} Queue;

// Create the queue
Queue * createQueue(){
    Queue *q = (Queue*) malloc (sizeof(Queue));
    q -> front = -1;
    q -> rear = -1;
    return q;
}

int isEmpty(Queue *q) {
    return (q -> front == -1);
}

int isFull(Queue *q) {
    return ((q -> rear +1) % SIZE == q -> front);
}
```



```
case 1:
    printf("Enter value to insert: ");
    scanf("%d", &value);
    insert(q, value);
    break;

case 2:
    delete(q);
    break;

case 3:
    display(q);
    break;

case 4:
    free(q);
    return 0;

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

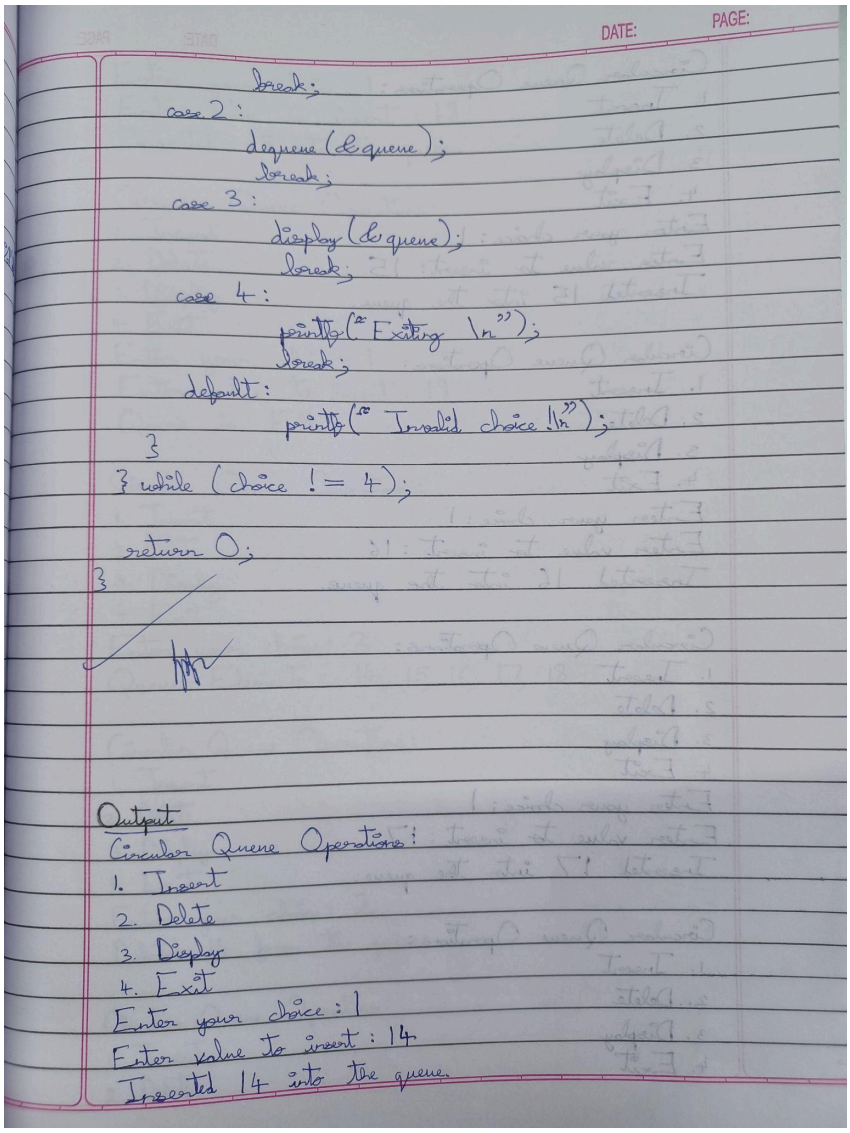


DATE: 21/10/24 PAGE:

Week - 3

☆ Circular Queue Program

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

#define MAX 5

typedef struct {
    int items[MAX];
    int front;
    int rear;
} CircularQueue;

void initQueue(CircularQueue *q) {
    q->front = -1;
    q->rear = -1;
}

int isFull(CircularQueue *q) {
    return (q->front == (q->rear + 1) % MAX);
}

int isEmpty(CircularQueue *q) {
    return (q->front == -1);
}
```



Code:

3a)

// Program for Queue Implementation using Arrays

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

#define SIZE 5

typedef struct Queue{
    int items[SIZE];
    int front;
    int rear;
} Queue;
```

```
// Create Queue
Queue* createQueue(){
    Queue *q = (Queue*)malloc(sizeof(Queue));

    q -> front = -1;
    q -> rear = -1;

    return q;
}



int isEmpty(Queue *q){
    return (q -> front == -1);
}

int isFull(Queue *q){
    return ((q -> rear + 1) % SIZE == q -> front);
}




// Insert Elements into Queue
void insert(Queue *q, int value){
    if(isFull(q)){
        printf("Queue Overflow! \n");
    }

    else{
        if(isEmpty(q)){
            q -> front = 0;
        }

        q -> rear = (q -> rear + 1);
        q -> items[q -> rear] = value;

        printf("Inserted %d into Queue. \n", value);
    }
}




//Dequeue or Delete the Elements
void delete(Queue *q){
    if(isEmpty(q)){
```

```
        printf("Queue Underflow! \n");
    }
    else{
        int removed_value = q -> items[q -> front];

        if(q -> front == q -> rear){                    //If the Queue has only 1 element
            q -> front = -1;
            q -> rear = -1;
        }
        else{
            q -> front = (q -> front + 1);
        }

        printf("Deleted %d from Queue. \n", removed_value);
    }
}




// Display the Queue
void display(Queue *q){
    if(isEmpty(q)){
        printf("Queue is Empty. \n");
    }
    else{
        int i = q -> front;

        printf("Queue elements: ");

        while(1){
            printf("%d, ", q -> items[i]);

            if(i == q -> rear){
                break;
            }

            i = (i + 1);
        }

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

```
// Main function
int main(){
   Queue *q = createQueue();
   int choice, value;

   while(1){
      printf(" 1. Insert \n 2. Delete \n 3. Display \n 4. Exit \n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch(choice){
         case 1:
            printf("Enter value to insert: ");
            scanf("%d", &value);
            insert(q, value);
            break;

         case 2:
            delete(q);
            break;

         case 3:
            display(q);
            break;

         case 4:
            free(q);
            return 0;

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

   return 0;
}
```

3b)

```
// Circular Queue Program

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


#define MAX 5
```

```
typedef struct{
   int items[MAX];
   int front;
   int rear;
}CircularQueue;


void initQueue(CircularQueue *q){
   q -> front = -1;
   q -> rear = -1;
}


int isFull(CircularQueue *q){
   return((q -> front) == ((q -> rear + 1) % MAX));
}

int isEmpty(CircularQueue *q){
   return(q -> front == -1);
}



void enqueue(CircularQueue *q, int value){
   if(isFull(q)){
      printf("Queue is Full! \n");
      return;
   }

   if(isEmpty(q)){
      q -> front = 0;
   }

   q -> rear = (q -> rear + 1) % MAX;
   q -> items[q -> rear] = value;
   printf("Inserted %d into the queue. \n", value);
}



void dequeue(CircularQueue *q){
   if(isEmpty(q)){
      printf("Queue is Empty! \n");
      return;
   }

   int deletedValue = q -> items[q -> front];
```

```
    printf("Deleted %d from the queue. \n", deletedValue);

    if(q -> front == q -> rear){
        q -> front = -1;
        q -> rear = -1;
    }
    else{
        q -> front = (q -> front + 1) % MAX;
    }
}




void display(CircularQueue *q){
    if(isEmpty(q)){
        printf("Queue is empty! \n");
        return;
    }

    printf("Queue Elements: ");

    for(int i = q -> front; i != q -> rear; i = (i + 1) % MAX){
        printf("%d, ", q -> items[i]);
    }

    printf("%d \n", q -> items[q -> rear]);
}




int main(){
    CircularQueue queue;
    initQueue(&queue);

    int choice, value;

    do{
        printf("\n Circular Queue Operations: \n");
        printf("1. Insert \n");
        printf("2. Delete \n");
        printf("3. Display \n");
        printf("4. Exit \n");

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

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

            case 2:
                dequeue(&queue);
                break;

            case 3:
                display(&queue);
                break;

            case 4:
                printf("Exiting \n");
                break;

            default:
                printf("Invalid choice! \n");
        }
    } while(choice != 4);

    return 0;
}
```

Output:

3a)

```
 1. Insert
 2. Delete
 3. Display
 4. Exit
Enter your choice: 3
Queue is Empty.
 1. Insert
 2. Delete
 3. Display
 4. Exit
Enter your choice: 1
Enter value to insert: 12
Inserted 12 into Queue.
 1. Insert
 2. Delete
 3. Display
 4. Exit
Enter your choice: 1
Enter value to insert: 15
Inserted 15 into Queue.
 1. Insert
 2. Delete
 3. Display
 4. Exit
Enter your choice: 3
Queue elements: 12, 15,
```

```
 1. Insert
 2. Delete
 3. Display
 4. Exit
Enter your choice: 2
Deleted 12 from Queue.
 1. Insert
 2. Delete
 3. Display
 4. Exit
Enter your choice: 3
Queue elements: 15,
 1. Insert
 2. Delete
 3. Display
 4. Exit
Enter your choice: 2
Deleted 15 from Queue.
 1. Insert
 2. Delete
 3. Display
 4. Exit
Enter your choice: 3
Queue is Empty.
```

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

3b)

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

 Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 15
Inserted 15 into the queue.

 Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 16
Inserted 16 into the queue.
```

```
 Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 17
Inserted 17 into the queue.

 Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 18
Inserted 18 into the queue.

 Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 19
Inserted 19 into the queue.
```

```
 Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 20
Queue is Full!

 Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue Elements: 15, 16, 17, 18, 19

 Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted 15 from the queue.
```

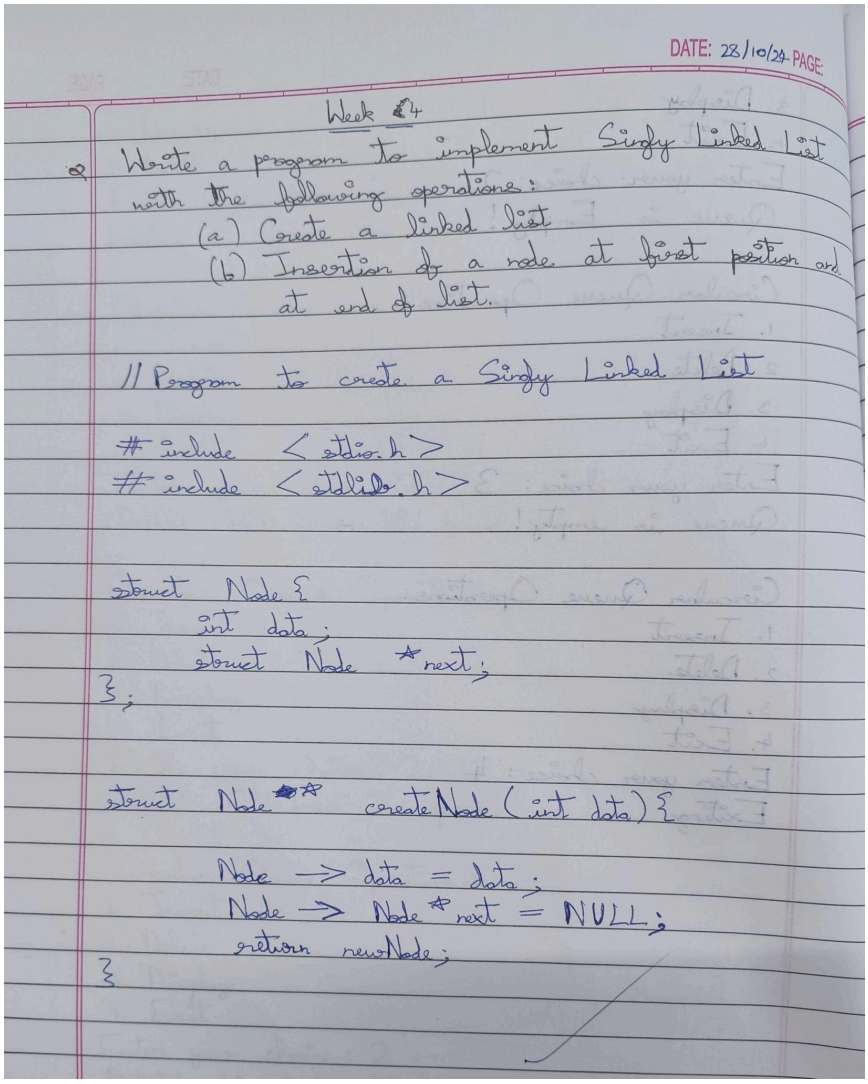
```
 Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue Elements: 16, 17, 18, 19

 Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 23
Inserted 23 into the queue.

 Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue Elements: 16, 17, 18, 19, 23
```

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

Lab Program 4

Write a program to Implement Singly Linked List with following operations:
a) Create a linked list.
b) Insertion of a node at first position, at any position and at end of list.
c) Display the contents of the linked list.

Screenshot of Observation:

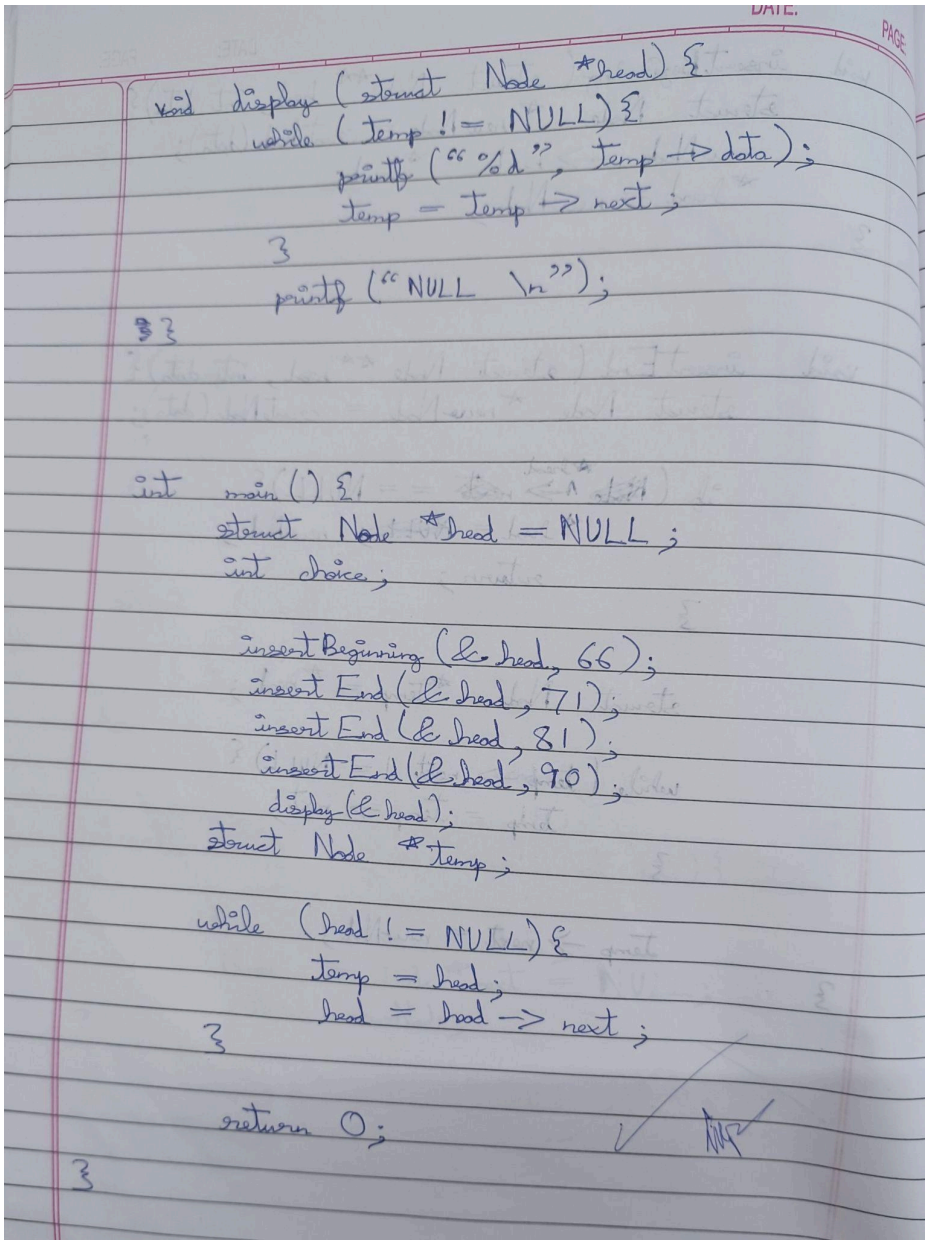


DATE: 28/10/24 PAGE:

Week 4

Q Write a program to implement Singly Linked List with the following operations:
(a) Create a linked list
(b) Insertion of a node at first position and at end of list.

```
// Program to create a Singly Linked List

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

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

struct Node** createNode(int data) {

    Node -> data = data;
    Node -> Node *next = NULL;
    return newNode;
}
```



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


int main () {
    struct Node *head = NULL;
    int choice;

    insertBeginning (&head, 66);
    insertEnd (&head, 71);
    insertEnd (&head, 81);
    insertEnd (&head, 90);
    display (&head);
    struct Node *temp;

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

    return 0;
}
```

Code:

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


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


struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
```

```
    newNode->next = NULL;
    return newNode;
}


struct Node* createLinkedList() {
    int data;
    struct Node *head = NULL, *temp = NULL;
    char choice;

    do {
        printf("Enter data for new node: ");
        scanf("%d", &data);

        struct Node* newNode = createNode(data);


        if (head == NULL) {
            head = newNode;
        } else {
            temp->next = newNode;
        }
        temp = newNode;

        printf("Do you want to add another node (y/n)? ");
        scanf(" %c", &choice);
    } while (choice == 'y' || choice == 'Y');

    return head;
}


struct Node* insertAtFirst(struct Node* head, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = head;
    head = newNode;
    return head;
}


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

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

```
    }
    temp->next = newNode;

    return head;
}


struct Node* insertAtPosition(struct Node* head, int data, int position) {
    if (position <= 0) {
        printf("Invalid position! Insertion failed.\n");
        return head;
    }

    struct Node* newNode = createNode(data);


    if (position == 1) {
        newNode->next = head;
        head = newNode;
        return head;
    }

    struct Node* temp = head;
    int count = 1;


    while (temp != NULL && count < position - 1) {
        temp = temp->next;
        count++;
    }


    if (temp == NULL) {
        printf("Position is greater than the length of the list.\n");
        free(newNode);
        return head;
    }


    newNode->next = temp->next;
    temp->next = newNode;

    return head;
}


void displayLinkedList(struct Node* head) {
    if (head == NULL) {
        printf("The list is empty.\n");
```

```
        return;
    }

    struct Node* temp = head;
    printf("Linked List: ");
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node* head = NULL;
    int choice, data, position;

    while (1) {
        printf("\nLinked List Operations:\n");
        printf("1. Create a linked list\n");
        printf("2. Insert at first position\n");
        printf("3. Insert at any position\n");
        printf("4. Insert at end position\n");
        printf("5. Display linked list\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                head = createLinkedList();
                printf("Linked list created successfully!\n");
                break;
            case 2:
                printf("Enter data to insert at the first position: ");
                scanf("%d", &data);
                head = insertAtFirst(head, data);
                printf("Node inserted at the first position.\n");
                break;
            case 3:
                printf("Enter data to insert: ");
                scanf("%d", &data);
                printf("Enter position to insert at: ");
                scanf("%d", &position);
                head = insertAtPosition(head, data, position);
                printf("Node inserted at position %d.\n", position);
                break;
            case 4:
                printf("Enter data to insert at the end: ");
                scanf("%d", &data);
```

```
            head = insertAtEnd(head, data);
            printf("Node inserted at the end of the list.\n");
            break;
        case 5:
            displayLinkedList(head);
            break;
        case 6:
            printf("Exiting the program.\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}
```

Output:

```
Linked List Operations:
1. Create a linked list
2. Insert at first position
3. Insert at any position
4. Insert at end position
5. Display linked list
6. Exit
Enter your choice: 1
Enter data for new node: 12
Do you want to add another node (y/n)? y
Enter data for new node: 15
Do you want to add another node (y/n)? y
Enter data for new node: 18
Do you want to add another node (y/n)? n
Linked list created successfully!

Linked List Operations:
1. Create a linked list
2. Insert at first position
3. Insert at any position
4. Insert at end position
5. Display linked list
6. Exit
Enter your choice: 5
Linked List: 12 -> 15 -> 18 -> NULL
```

```
Linked List Operations:
1. Create a linked list
2. Insert at first position
3. Insert at any position
4. Insert at end position
5. Display linked list
6. Exit
Enter your choice: 2
Enter data to insert at the first position: 21
Node inserted at the first position.

Linked List Operations:
1. Create a linked list
2. Insert at first position
3. Insert at any position
4. Insert at end position
5. Display linked list
6. Exit
Enter your choice: 3
Enter data to insert: 54
Enter position to insert at: 2
Node inserted at position 2.
```

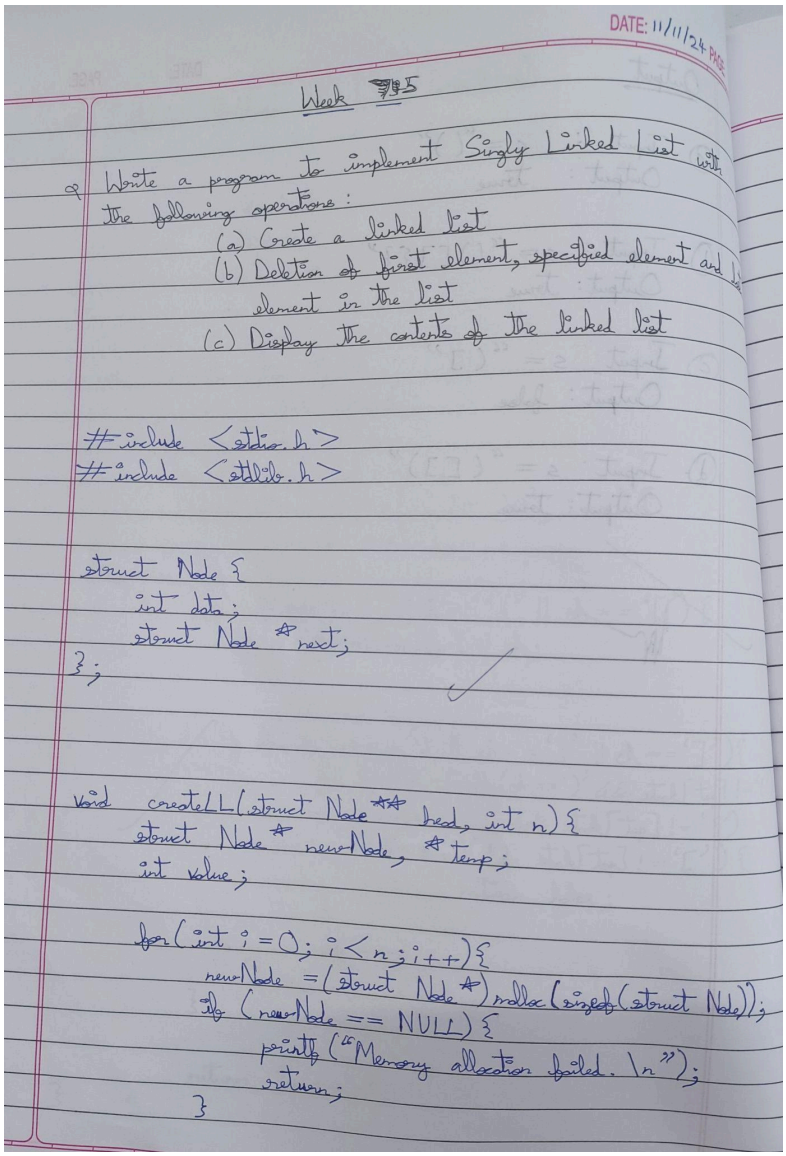
```
Linked List Operations:
1. Create a linked list
2. Insert at first position
3. Insert at any position
4. Insert at end position
5. Display linked list
6. Exit
Enter your choice: 5
Linked List: 21 -> 54 -> 12 -> 15 -> 18 -> NULL

Linked List Operations:
1. Create a linked list
2. Insert at first position
3. Insert at any position
4. Insert at end position
5. Display linked list
6. Exit
Enter your choice: 4
Enter data to insert at the end: 76
Node inserted at the end of the list.
```

```
Linked List Operations:
1. Create a linked list
2. Insert at first position
3. Insert at any position
4. Insert at end position
5. Display linked list
6. Exit
Enter your choice: 5
Linked List: 21 -> 54 -> 12 -> 15 -> 18 -> 76 -> NULL

Linked List Operations:
1. Create a linked list
2. Insert at first position
3. Insert at any position
4. Insert at end position
5. Display linked list
6. Exit
Enter your choice: 6
Exiting the program.
```

Lab Program 5

Write a program 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.

Screenshot of Observation:

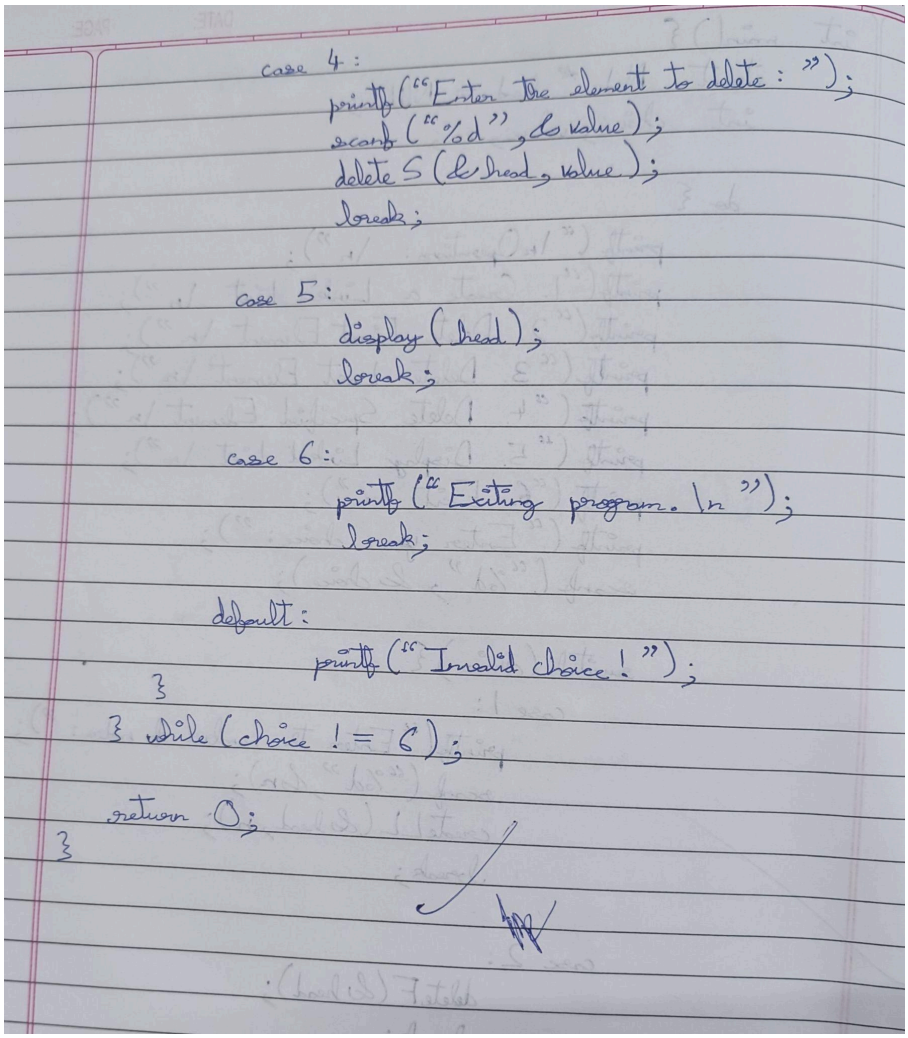


DATE: 11/11/24 PAGE

Week 5

Q Write a program to implement Singly Linked List with the 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

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

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

void createLL(struct Node **head, int n) {
    struct Node *newNode, *temp;
    int value;

    for(int i=0; i<n; i++){
        newNode = (struct Node *) malloc(sizeof(struct Node));
        if (newNode == NULL) {
            printf("Memory allocation failed.\n");
            return;
        }
```



Code:

```
// Singly Linked List


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


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



void createLL(struct Node** head, int n){
    struct Node* newNode, *temp;
    int value;
```

```
    for(int i = 0; i < n; i++){
        newNode = (struct Node*) malloc (sizeof(struct Node));

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

        printf("Enter value for node %d: ", i+1);
        scanf("%d", &value);

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

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



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

    struct Node* temp = *head;
    *head = (*head) -> next;

    free(temp);
    printf("First element deleted. \n");
}



void deleteL(struct Node** head){
    if(*head == NULL){
        printf("The list is already empty. \n");
        return;
```

```
    }

    struct Node* temp = *head, *prev = NULL;

    if(temp -> next == NULL){
        free(temp);
        *head = NULL;
        printf("Last element deleted. \n");
        return;
    }

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

    prev -> next = NULL;
    free(temp);
    printf("Last element deleted. \n");
}



void deleteS(struct Node** head, int value){
    if(*head == NULL){
        printf("The list is already empty. \n");
        return;
    }

    struct Node* temp = *head, *prev = NULL;

    if(temp != NULL && temp -> data == value){
        *head = temp -> next;
        free(temp);
        printf("Element %d deleted. \n", value);
        return;
    }

    while(temp != NULL && temp -> data != value){
        prev = temp;
        temp = temp -> next;
    }

    if(temp == NULL){
        printf("Element %d not found. \n", value);
        return;
    }
```

```
    prev -> next = temp -> next;
    free(temp);
    printf("Element %d deleted. \n", value);
}



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

    struct Node* temp = head;

    printf("Linked List: ");

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

    printf("NULL \n");

}




int main(){
    struct Node* head = NULL;
    int choice, value, n;

    do{
        printf("\nOperations: \n");
        printf("1. Create a Linked List \n");
        printf("2. Delete First Element \n");
        printf("3. Delete Last Element \n");
        printf("4. Delete Specified Element \n");
        printf("5. Display Linked List \n");
        printf("6. Exit \n");

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

```
        switch(choice){
            case 1:
                printf("Enter the number of nodes: ");
                scanf("%d", &n);
                createLL(&head, n);
                break;

            case 2:
                deleteF(&head);
                break;

            case 3:
                deleteL(&head);
                break;

            case 4:
                printf("Enter the element to delete: ");
                scanf("%d", &value);
                deleteS(&head, value);
                break;

            case 5:
                display(head);
                break;

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

            default:
                printf("Invalid choice!");
        }
    } while(choice != 6);

    return 0;
}
```

Output:

```
Operations:
1. Create a Linked List
2. Delete First Element
3. Delete Last Element
4. Delete Specified Element
5. Display Linked List
6. Exit
Enter your choice: 1
Enter the number of nodes: 5
Enter value for node 1: 12
Enter value for node 2: 13
Enter value for node 3: 14
Enter value for node 4: 15
Enter value for node 5: 21

Operations:
1. Create a Linked List
2. Delete First Element
3. Delete Last Element
4. Delete Specified Element
5. Display Linked List
6. Exit
Enter your choice: 5
Linked List: 12 -> 13 -> 14 -> 15 -> 21 -> NULL
```

```
1. Create a Linked List
2. Delete First Element
3. Delete Last Element
4. Delete Specified Element
5. Display Linked List
6. Exit
Enter your choice: 2
First element deleted.

Operations:
1. Create a Linked List
2. Delete First Element
3. Delete Last Element
4. Delete Specified Element
5. Display Linked List
6. Exit
Enter your choice: 5
Linked List: 13 -> 14 -> 15 -> 21 -> NULL

Operations:
1. Create a Linked List
2. Delete First Element
3. Delete Last Element
4. Delete Specified Element
5. Display Linked List
6. Exit
Enter your choice: 3
Last element deleted.
```

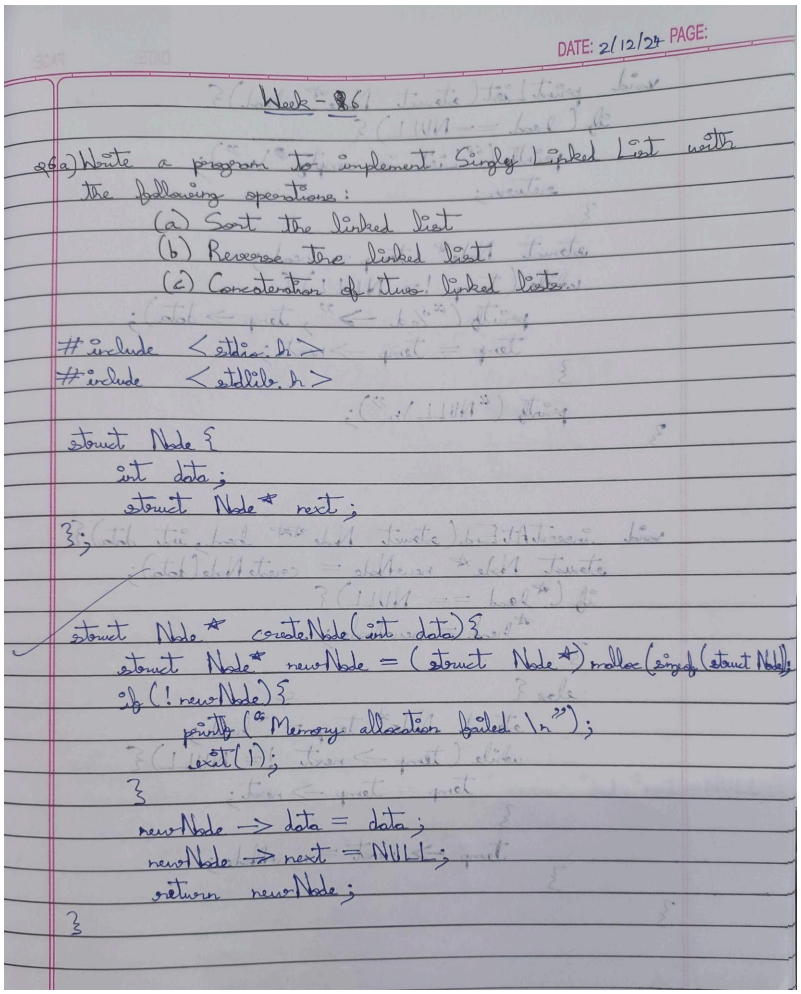
```
Operations:
1. Create a Linked List
2. Delete First Element
3. Delete Last Element
4. Delete Specified Element
5. Display Linked List
6. Exit
Enter your choice: 5
Linked List: 13 -> 14 -> 15 -> NULL

Operations:
1. Create a Linked List
2. Delete First Element
3. Delete Last Element
4. Delete Specified Element
5. Display Linked List
6. Exit
Enter your choice: 4
Enter the element to delete: 14
Element 14 deleted.
```

```
Operations:
1. Create a Linked List
2. Delete First Element
3. Delete Last Element
4. Delete Specified Element
5. Display Linked List
6. Exit
Enter your choice: 5
Linked List: 13 -> 15 -> NULL

Operations:
1. Create a Linked List
2. Delete First Element
3. Delete Last Element
4. Delete Specified Element
5. Display Linked List
6. Exit
Enter your choice: 6
Exiting program.
```

Lab Program 6

6a) Write a program to Implement Singly Linked List with the following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists.
6b) Write a program to Implement Singly Linked List to simulate Stack & Queue Operations.

Screenshot of Observation:

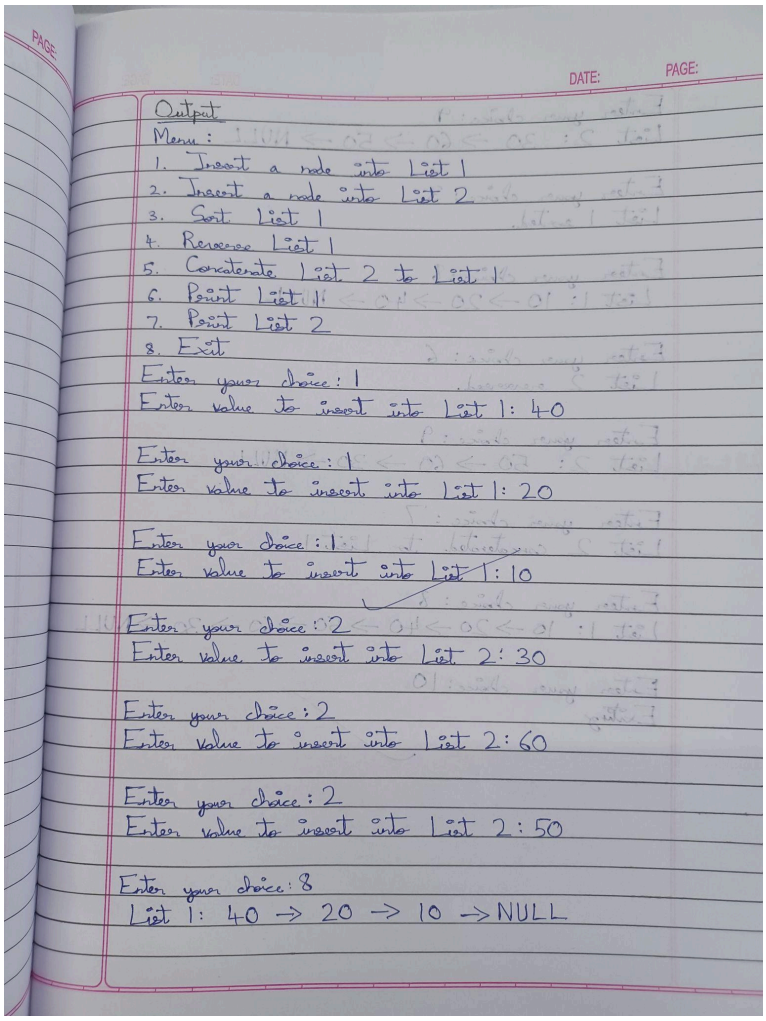


DATE: 2/12/24 PAGE:

Week - 8

8(a) Write a program to implement Singly Linked List with the following operations:
(a) Sort the linked list
(b) Reverse the linked list
(c) Concatenation of two linked lists

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

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

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*) malloc (sizeof(struct Node));
    if (!newNode) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode -> data = data;
    newNode -> next = NULL;
    return newNode;
}
```



Output

Menu :
1. Insert a node into List 1
2. Insert a node into List 2
3. Sort List 1
4. Reverse List 1
5. Concatenate List 2 to List 1
6. Print List 1
7. Print List 2
8. Exit

Enter your choice : 1
Enter value to insert into List 1: 40

Enter your choice : 1
Enter value to insert into List 1: 20

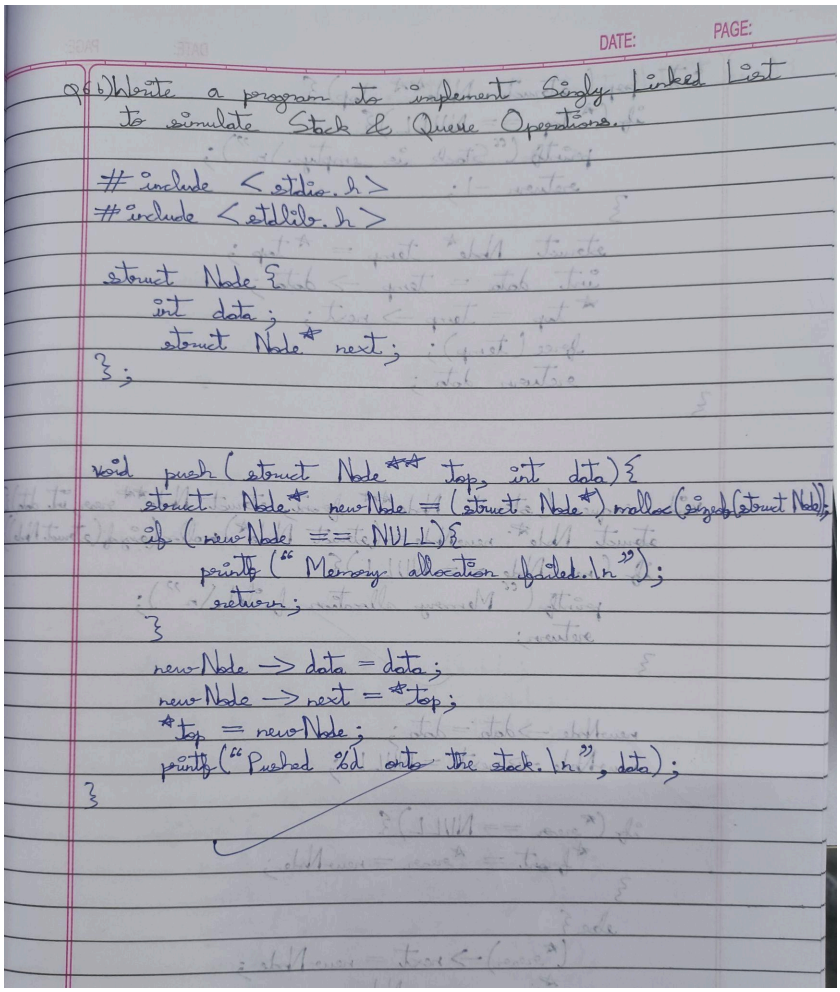
Enter your choice : 1
Enter value to insert into List 1: 10

Enter your choice : 2
Enter value to insert into List 2: 30

Enter your choice : 2
Enter value to insert into List 2: 60

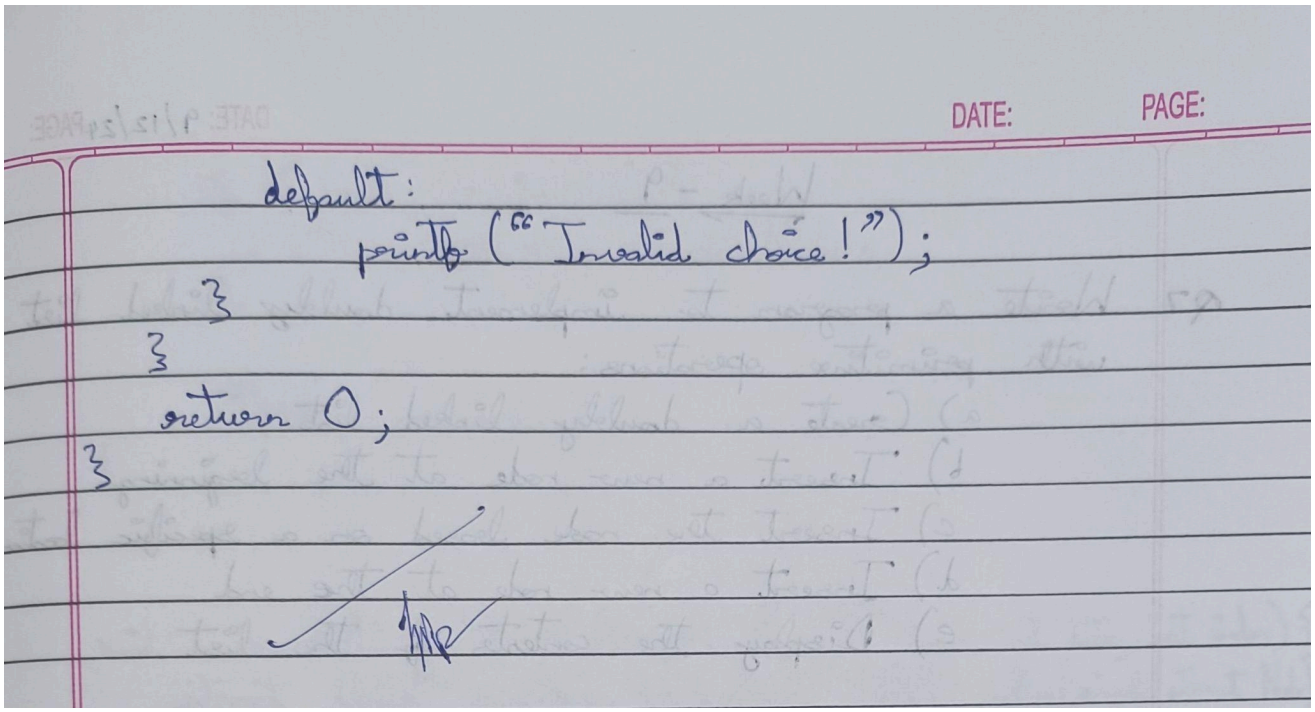
Enter your choice : 2
Enter value to insert into List 2: 50

Enter your choice: 8
List 1: 40 -> 20 -> 10 -> NULL



Q6 b) Write a program to implement Singly linked List to simulate Stack & Queue Operations.

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

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

void push(struct Node** top, 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 = *top;
    *top = newNode;
    printf("Pushed %d onto the stack.\n", data);
}
```



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

Code:

6a)

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


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


struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}


struct Node* createLinkedList() {
    int data;
    struct Node *head = NULL, *temp = NULL;
    char choice;

    do {
        printf("Enter data for new node: ");
        scanf("%d", &data);

        struct Node* newNode = createNode(data);


        if (head == NULL) {
            head = newNode;
        } else {
            temp->next = newNode;
        }
        temp = newNode;

        printf("Do you want to add another node (y/n)? ");
        scanf(" %c", &choice);
    } while (choice == 'y' || choice == 'Y');

    return head;
}


void displayLinkedList(struct Node* head) {
    if (head == NULL) {
        printf("The list is empty.\n");
        return;
```

```
    }

    struct Node* temp = head;
    printf("Linked List: ");
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}


void sortLinkedList(struct Node* head) {
    if (head == NULL) {
        printf("The list is empty, cannot sort.\n");
        return;
    }

    struct Node *i, *j;
    int temp;
    for (i = head; i != NULL; i = i->next) {
        for (j = i->next; j != NULL; j = j->next) {
            if (i->data > j->data) {
                // Swap data
                temp = i->data;
                i->data = j->data;
                j->data = temp;
            }
        }
    }
    printf("List sorted successfully.\n");
}


struct Node* reverseLinkedList(struct Node* head) {
    struct Node *prev = NULL, *curr = head, *next = NULL;
    while (curr != NULL) {
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
    head = prev;
    printf("List reversed successfully.\n");
    return head;
}


struct Node* concatenateLists(struct Node* head1, struct Node* head2) {
```

```
    if (head1 == NULL) return head2;
    if (head2 == NULL) return head1;

    struct Node* temp = head1;
    while (temp->next != NULL) {
        temp = temp->next;
    }


    temp->next = head2;

    printf("Lists concatenated successfully.\n");
    return head1;
}

int main() {
    struct Node *head1 = NULL, *head2 = NULL;
    int choice, data;

    while (1) {
        printf("\nLinked List Operations:\n");
        printf("1. Create first linked list\n");
        printf("2. Create second linked list\n");
        printf("3. Display first linked list\n");
        printf("4. Display second linked list\n");
        printf("5. Sort first linked list\n");
        printf("6. Reverse first linked list\n");
        printf("7. Concatenate two linked lists\n");
        printf("8. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                head1 = createLinkedList();
                printf("First linked list created successfully!\n");
                break;
            case 2:
                head2 = createLinkedList();
                printf("Second linked list created successfully!\n");
                break;
            case 3:
                displayLinkedList(head1);
                break;
            case 4:
                displayLinkedList(head2);
                break;
            case 5:
                sortLinkedList(head1);
```

```
            break;
        case 6:
            head1 = reverseLinkedList(head1);
            break;
        case 7:
            head1 = concatenateLists(head1, head2);
            break;
        case 8:
            printf("Exiting the program.\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}
```

6b)

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


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


struct Node* createNode(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = NULL;
   return newNode;
}


void push(struct Node** top, int data) {
   struct Node* newNode = createNode(data);
   newNode->next = *top;
   *top = newNode;
   printf("Pushed %d onto stack.\n", data);
}


int pop(struct Node** top) {
```

```
    if (*top == NULL) {
        printf("Stack underflow!\n");
        return -1;
    }
    struct Node* temp = *top;
    int data = temp->data;
    *top = temp->next;
    free(temp);
    return data;
}

void displayStack(struct Node* top) {
    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("\n");
}

void enqueue(struct Node** front, struct Node** rear, int data) {
    struct Node* newNode = createNode(data);
    if (*rear == NULL) {
        *front = *rear = newNode;
        printf("Enqueued %d into queue.\n", data);
        return;
    }
    (*rear)->next = newNode;
    *rear = newNode;
    printf("Enqueued %d into queue.\n", data);
}

int dequeue(struct Node** front, struct Node** rear) {
    if (*front == NULL) {
        printf("Queue underflow!\n");
        return -1;
    }
    struct Node* temp = *front;
    int data = temp->data;
    *front = (*front)->next;
    if (*front == NULL) {
```

```
        *rear = NULL;
    }
    free(temp);
    return data;
}


void displayQueue(struct Node* front) {
    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("\n");
}

int main() {
    struct Node* stackTop = NULL;
    struct Node* queueFront = NULL;
    struct Node* queueRear = NULL;
    int choice, data;

    while (1) {
        printf("\nSelect Operation:\n");
        printf("1. Push (Stack)\n");
        printf("2. Pop (Stack)\n");
        printf("3. Display Stack\n");
        printf("4. Enqueue (Queue)\n");
        printf("5. Dequeue (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 onto stack: ");
                scanf("%d", &data);
                push(&stackTop, data);
                break;
            case 2:
                data = pop(&stackTop);
                if (data != -1) {
                    printf("Popped %d from stack.\n", data);
```

```
            }
            break;
        case 3:
            displayStack(stackTop);
            break;
        case 4:
            printf("Enter data to enqueue into queue: ");
            scanf("%d", &data);
            enqueue(&queueFront, &queueRear, data);
            break;
        case 5:
            data = dequeue(&queueFront, &queueRear);
            if (data != -1) {
                printf("Dequeued %d from queue.\n", data);
            }
            break;
        case 6:
            displayQueue(queueFront);
            break;
        case 7:
            printf("Exiting the program.\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
    }
  }

  return 0;
}
```

Output:

6a)

```
Linked List Operations:
1. Create first linked list
2. Create second linked list
3. Display first linked list
4. Display second linked list
5. Sort first linked list
6. Reverse first linked list
7. Concatenate two linked lists
8. Exit
Enter your choice: 1
Enter data for new node: 15
Do you want to add another node (y/n)? y
Enter data for new node: 12
Do you want to add another node (y/n)? y
Enter data for new node: 16
Do you want to add another node (y/n)? n
First linked list created successfully!
```

```
Linked List Operations:
1. Create first linked list
2. Create second linked list
3. Display first linked list
4. Display second linked list
5. Sort first linked list
6. Reverse first linked list
7. Concatenate two linked lists
8. Exit
Enter your choice: 2
Enter data for new node: 54
Do you want to add another node (y/n)? y
Enter data for new node: 35
Do you want to add another node (y/n)? y
Enter data for new node: 91
Do you want to add another node (y/n)? n
Second linked list created successfully!
```

```
Linked List Operations:
1. Create first linked list
2. Create second linked list
3. Display first linked list
4. Display second linked list
5. Sort first linked list
6. Reverse first linked list
7. Concatenate two linked lists
8. Exit
Enter your choice: 3
Linked List: 15 -> 12 -> 16 -> NULL

Linked List Operations:
1. Create first linked list
2. Create second linked list
3. Display first linked list
4. Display second linked list
5. Sort first linked list
6. Reverse first linked list
7. Concatenate two linked lists
8. Exit
Enter your choice: 4
Linked List: 54 -> 35 -> 91 -> NULL
```

```
Linked List Operations:
1. Create first linked list
2. Create second linked list
3. Display first linked list
4. Display second linked list
5. Sort first linked list
6. Reverse first linked list
7. Concatenate two linked lists
8. Exit
Enter your choice: 5
List sorted successfully.

Linked List Operations:
1. Create first linked list
2. Create second linked list
3. Display first linked list
4. Display second linked list
5. Sort first linked list
6. Reverse first linked list
7. Concatenate two linked lists
8. Exit
Enter your choice: 3
Linked List: 12 -> 15 -> 16 -> NULL
```

```
Linked List Operations:
1. Create first linked list
2. Create second linked list
3. Display first linked list
4. Display second linked list
5. Sort first linked list
6. Reverse first linked list
7. Concatenate two linked lists
8. Exit
Enter your choice: 6
List reversed successfully.

Linked List Operations:
1. Create first linked list
2. Create second linked list
3. Display first linked list
4. Display second linked list
5. Sort first linked list
6. Reverse first linked list
7. Concatenate two linked lists
8. Exit
Enter your choice: 3
Linked List: 16 -> 15 -> 12 -> NULL
```

```
Linked List Operations:
1. Create first linked list
2. Create second linked list
3. Display first linked list
4. Display second linked list
5. Sort first linked list
6. Reverse first linked list
7. Concatenate two linked lists
8. Exit
Enter your choice: 7
Lists concatenated successfully.

Linked List Operations:
1. Create first linked list
2. Create second linked list
3. Display first linked list
4. Display second linked list
5. Sort first linked list
6. Reverse first linked list
7. Concatenate two linked lists
8. Exit
Enter your choice: 3
Linked List: 16 -> 15 -> 12 -> 54 -> 35 -> 91 -> NULL
```

```
Linked List Operations:
1. Create first linked list
2. Create second linked list
3. Display first linked list
4. Display second linked list
5. Sort first linked list
6. Reverse first linked list
7. Concatenate two linked lists
8. Exit
Enter your choice: 8
Exiting the program.
```

6b)

```
Select Operation:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 1
Enter data to push onto stack: 14
Pushed 14 onto stack.

Select Operation:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 1
Enter data to push onto stack: 15
Pushed 15 onto stack.
```

```
Select Operation:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 1
Enter data to push onto stack: 21
Pushed 21 onto stack.

Select Operation:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 3
Stack: 21 15 14
```

```
Select Operation:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 2
Popped 21 from stack.

Select Operation:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 3
Stack: 15 14
```

```
Select Operation:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 4
Enter data to enqueue into queue: 54
Enqueued 54 into queue.

Select Operation:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 4
Enter data to enqueue into queue: 65
Enqueued 65 into queue.
```

```
Select Operation:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 4
Enter data to enqueue into queue: 76
Enqueued 76 into queue.

Select Operation:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 6
Queue: 54 65 76
```

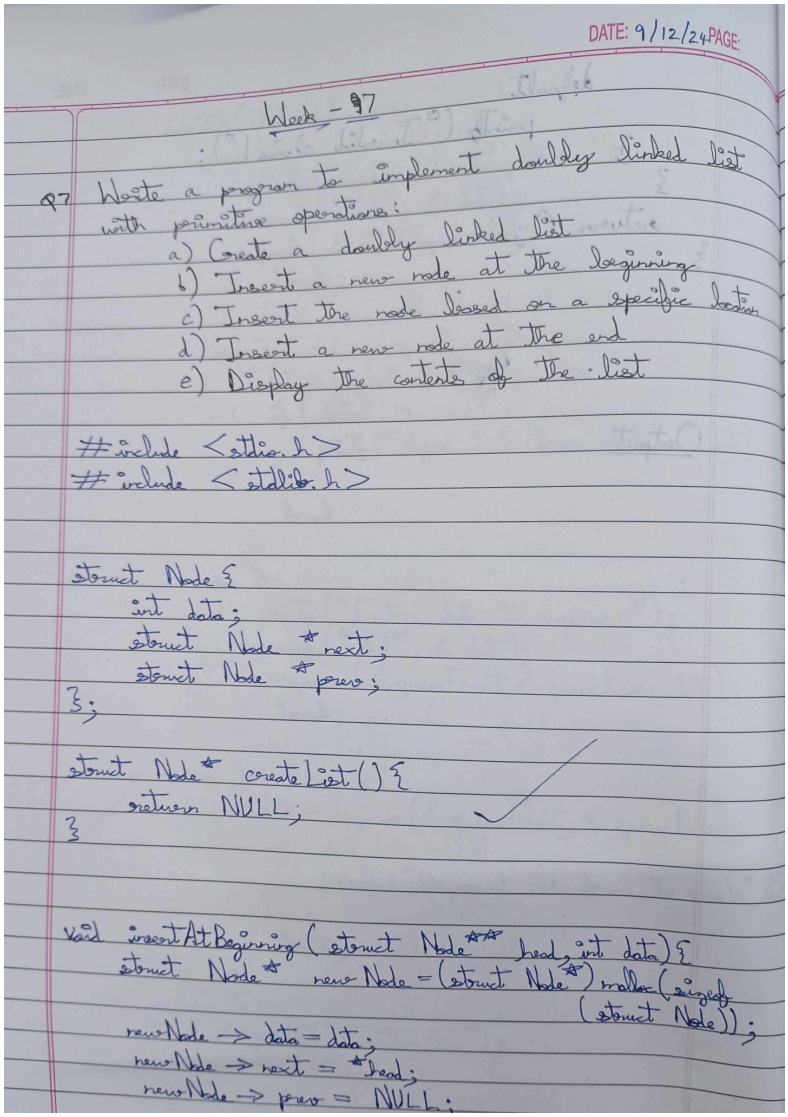
```
Select Operation:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 5
Dequeued 54 from queue.

Select Operation:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 6
Queue: 65 76
```

```
Select Operation:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 7
Exiting the program.
```

Lab Program 7

Write a program 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

Screenshot of Observation:

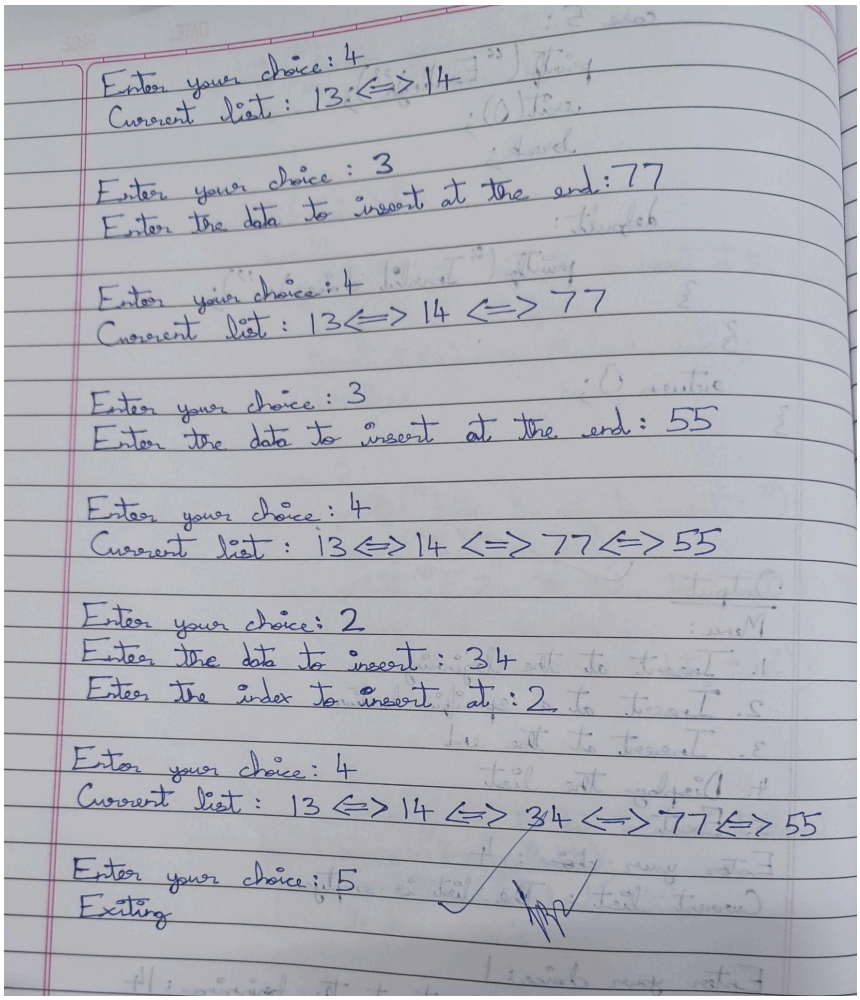


DATE: 9/12/24 PAGE:

Week - 7

Q7 Write a program to implement doubly linked list with primitive operations:
a) Create a doubly linked list
b) Insert a new node at the beginning
c) Insert the node based on a specific location
d) Insert a new node at the end
e) Display the contents of the list

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

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

struct Node* createList() {
    return NULL;
}

void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = (struct Node*) malloc(sizeof
                                (struct Node));
    newNode -> data = data;
    newNode -> next = *head;
    newNode -> prev = NULL;
```



Code:

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


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


struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}
```

```
void createList(struct Node** head) {
    *head = NULL;
    int n, data;
    printf("Enter the number of nodes: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        printf("Enter data for node %d: ", i + 1);
        scanf("%d", &data);
        struct Node* newNode = createNode(data);
        if (*head == NULL) {
            *head = newNode;
        } else {
            struct Node* temp = *head;
            while (temp->next != NULL) {
                temp = temp->next;
            }
            temp->next = newNode;
            newNode->prev = temp;
        }
    }
    printf("List created successfully.\n");
}


void insertLeft(struct Node** head, int targetValue, int data) {
    struct Node* temp = *head;
    while (temp != NULL && temp->data != targetValue) {
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Node with value %d not found in the list.\n", targetValue);
        return;
    }

    struct Node* newNode = createNode(data);
    newNode->next = temp;
    newNode->prev = temp->prev;

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

    printf("Node with value %d inserted to the left of node %d.\n", data, targetValue);
}
```

```
void deleteNode(struct Node** head, int value) {
    struct Node* temp = *head;


    if (temp == NULL) {
        printf("List is empty.\n");
        return;
    }


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


    if (temp == NULL) {
        printf("Node with value %d not found in the list.\n", value);
        return;
    }


    if (temp->prev == NULL && temp->next == NULL) {
        *head = NULL;
    }

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

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

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

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


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

```
    struct Node* temp = head;
    printf("List contents: ");
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    struct Node* head = NULL;
    int choice, data, targetValue;

    while (1) {
        printf("\nSelect operation:\n");
        printf("1. Create Doubly Linked List\n");
        printf("2. Insert a new node to the left of a node\n");
        printf("3. Delete a node by value\n");
        printf("4. Display the list\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createList(&head);
                break;
            case 2:
                printf("Enter the value of the node to insert before: ");
                scanf("%d", &targetValue);
                printf("Enter the data for the new node: ");
                scanf("%d", &data);
                insertLeft(&head, targetValue, data);
                break;
            case 3:
                printf("Enter the value of the node to delete: ");
                scanf("%d", &data);
                deleteNode(&head, data);
                break;
            case 4:
                displayList(head);
                break;
            case 5:
                printf("Exiting program.\n");
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
```

```
    }

    return 0;
}
```

Output:

```
Select operation:
1. Create Doubly Linked List
2. Insert a new node to the left of a node
3. Delete a node by value
4. Display the list
5. Exit
Enter your choice: 1
Enter the number of nodes: 5
Enter data for node 1: 34
Enter data for node 2: 23
Enter data for node 3: 12
Enter data for node 4: 56
Enter data for node 5: 75
List created successfully.

Select operation:
1. Create Doubly Linked List
2. Insert a new node to the left of a node
3. Delete a node by value
4. Display the list
5. Exit
Enter your choice: 4
List contents: 34 -> 23 -> 12 -> 56 -> 75 ->
```

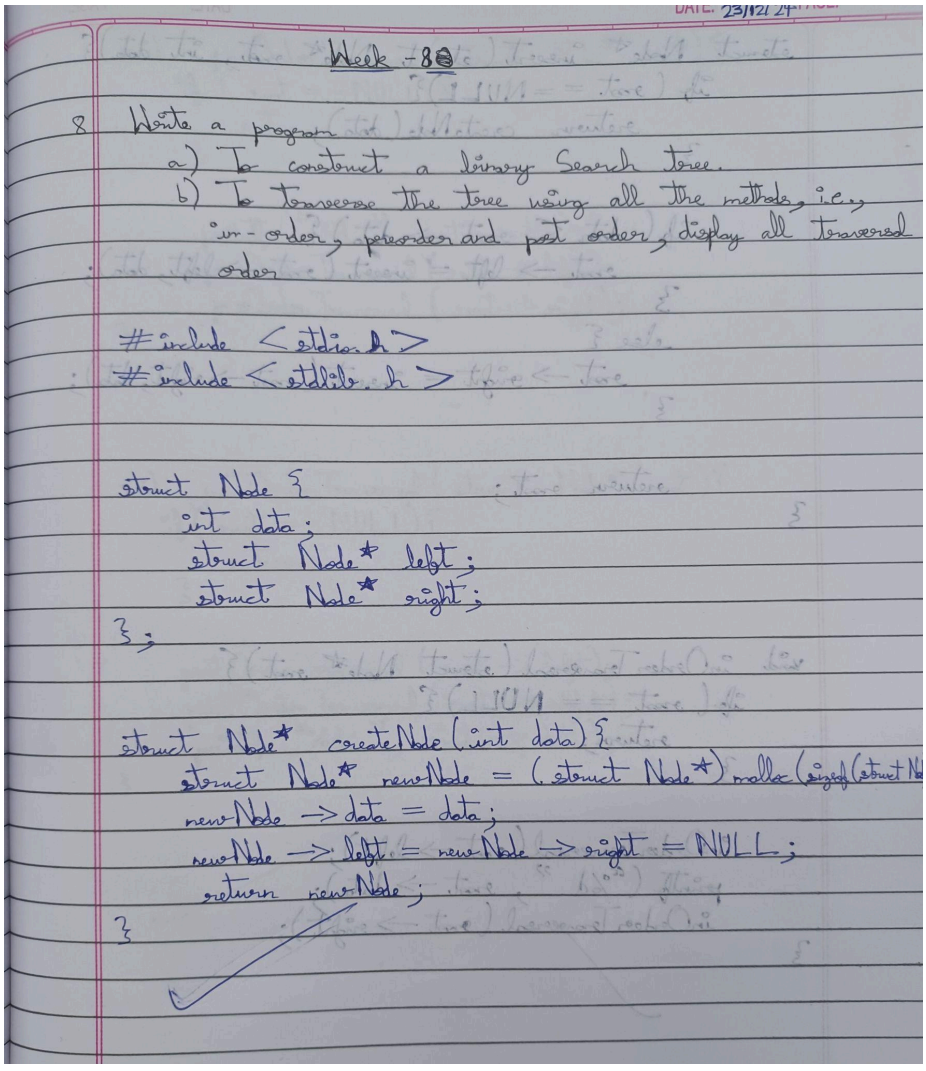
```
Select operation:
1. Create Doubly Linked List
2. Insert a new node to the left of a node
3. Delete a node by value
4. Display the list
5. Exit
Enter your choice: 2
Enter the value of the node to insert before: 56
Enter the data for the new node: 6
Node with value 6 inserted to the left of node 56.

Select operation:
1. Create Doubly Linked List
2. Insert a new node to the left of a node
3. Delete a node by value
4. Display the list
5. Exit
Enter your choice: 4
List contents: 34 -> 23 -> 12 -> 6 -> 56 -> 75 ->

Select operation:
1. Create Doubly Linked List
2. Insert a new node to the left of a node
3. Delete a node by value
4. Display the list
5. Exit
Enter your choice: 3
Enter the value of the node to delete: 23
Node with value 23 deleted.
```

```
Select operation:
1. Create Doubly Linked List
2. Insert a new node to the left of a node
3. Delete a node by value
4. Display the list
5. Exit
Enter your choice: 4
List contents: 34 -> 12 -> 6 -> 56 -> 75 ->

Select operation:
1. Create Doubly Linked List
2. Insert a new node to the left of a node
3. Delete a node by value
4. Display the list
5. Exit
Enter your choice: 5
Exiting program.
```

Lab Program 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.

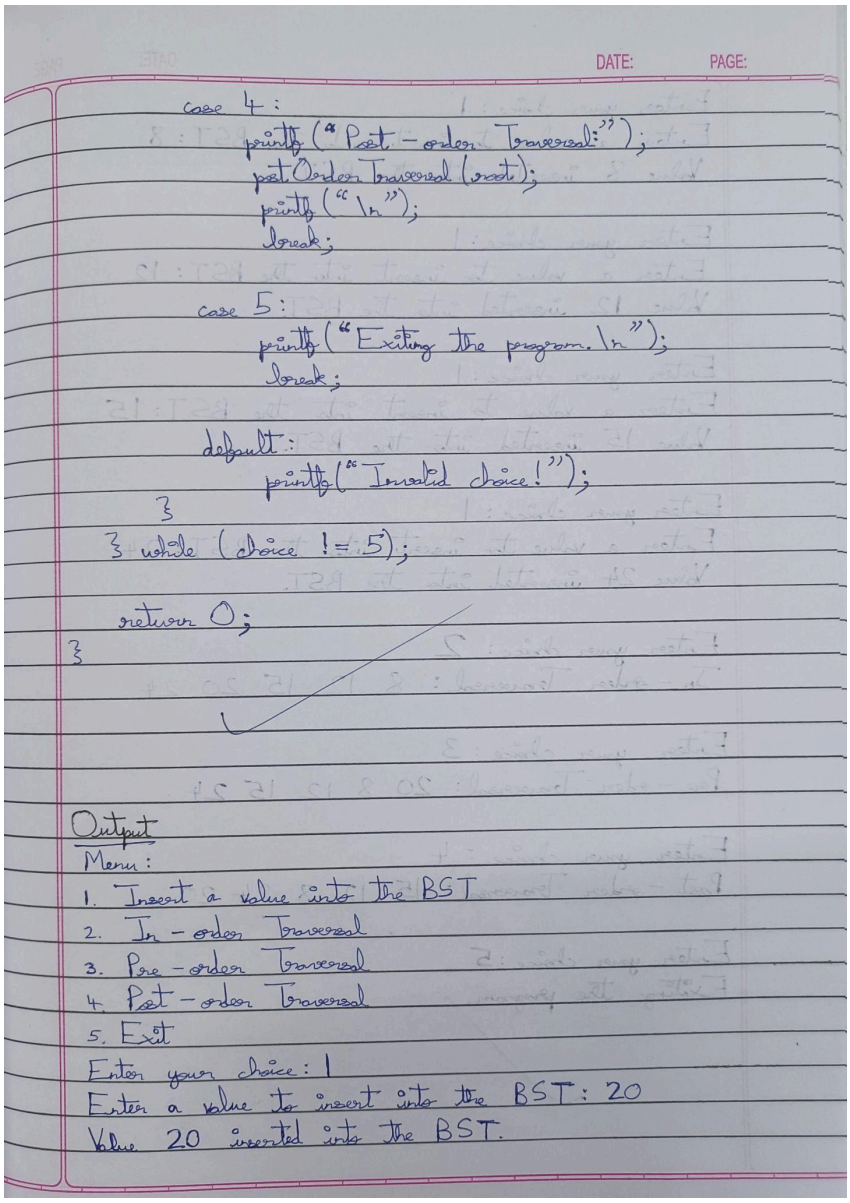
Screenshot of Observation:



Week - 8

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, display all traversed order

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

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

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*) malloc(sizeof(struct N
    newNode -> data = data;
    newNode -> left = newNode -> right = NULL;
    return newNode;
}
```



```
case 4:
    printf("Post-order Traversal:");
    postOrderTraversal(root);
    printf("\n");
    break;

case 5:
    printf("Exiting the program.\n");
    break;

default:
    printf("Invalid choice!");
}
} while (choice != 5);

return 0;
}
```

Output

```
Menu:
1. Insert a value into the BST
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter a value to insert into the BST: 20
Value 20 inserted into the BST.
```

Code:

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


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


struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
```

```
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}


struct Node* insert(struct Node* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }

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

    return root;
}


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


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


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


void displayTree(struct Node* root) {
```

```
    printf("\nIn-order traversal: ");
    inorderTraversal(root);

    printf("\nPre-order traversal: ");
    preorderTraversal(root);

    printf("\nPost-order traversal: ");
    postorderTraversal(root);
}

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

    while (1) {
        printf("\nBinary Search Tree Operations:\n");
        printf("1. Insert Node\n");
        printf("2. Display Tree (All Traversals)\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to insert: ");
                scanf("%d", &data);
                root = insert(root, data);
                printf("Node inserted.\n");
                break;
            case 2:
                displayTree(root);
                break;
            case 3:
                printf("Exiting the program.\n");
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

Output:

```
Binary Search Tree Operations:
1. Insert Node
2. Display Tree (All Traversals)
3. Exit
Enter your choice: 1
Enter data to insert: 23
Node inserted.

Binary Search Tree Operations:
1. Insert Node
2. Display Tree (All Traversals)
3. Exit
Enter your choice: 1
Enter data to insert: 15
Node inserted.

Binary Search Tree Operations:
1. Insert Node
2. Display Tree (All Traversals)
3. Exit
Enter your choice: 1
Enter data to insert: 12
Node inserted.
```

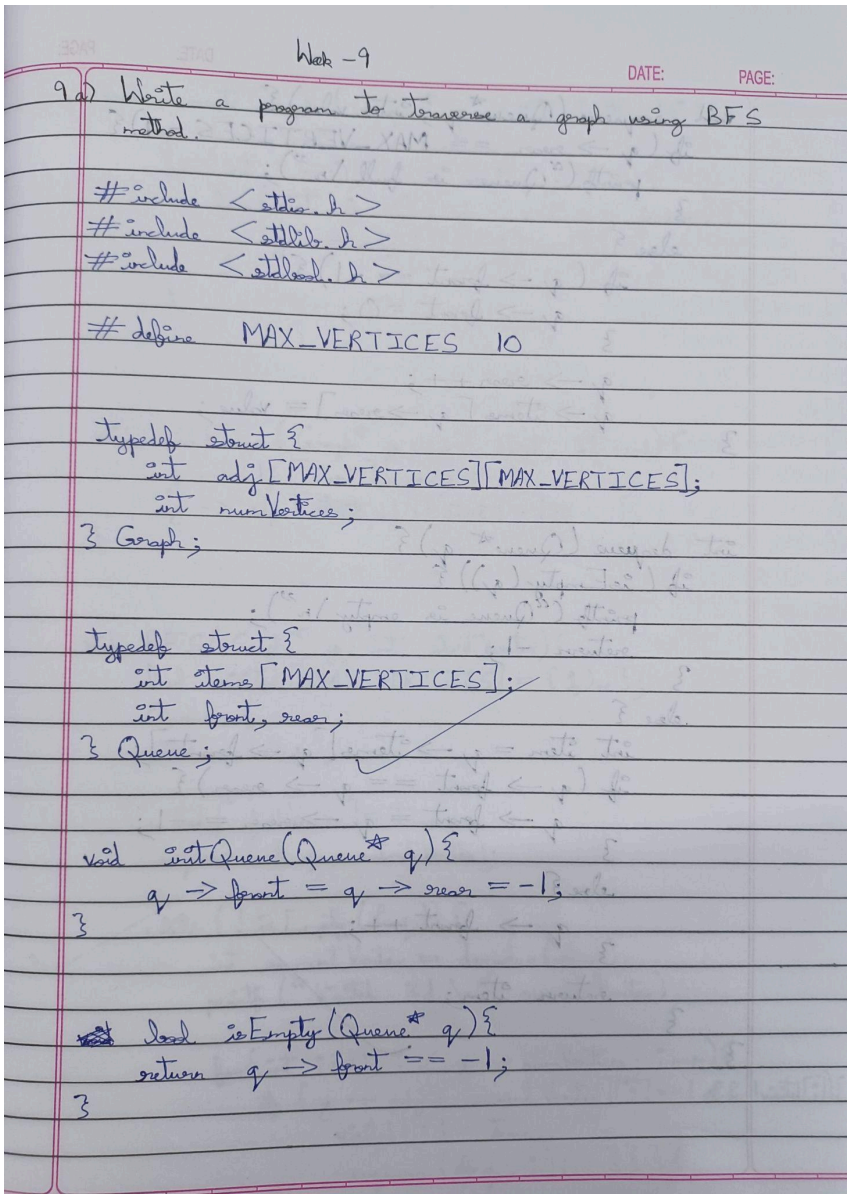
```
Binary Search Tree Operations:
1. Insert Node
2. Display Tree (All Traversals)
3. Exit
Enter your choice: 1
Enter data to insert: 18
Node inserted.

Binary Search Tree Operations:
1. Insert Node
2. Display Tree (All Traversals)
3. Exit
Enter your choice: 1
Enter data to insert: 25
Node inserted.

Binary Search Tree Operations:
1. Insert Node
2. Display Tree (All Traversals)
3. Exit
Enter your choice: 1
Enter data to insert: 24
Node inserted.
```

```
Binary Search Tree Operations:
1. Insert Node
2. Display Tree (All Traversals)
3. Exit
Enter your choice: 1
Enter data to insert: 30
Node inserted.

Binary Search Tree Operations:
1. Insert Node
2. Display Tree (All Traversals)
3. Exit
Enter your choice: 2

In-order traversal: 12 15 18 23 24 25 30
Pre-order traversal: 23 15 12 18 25 24 30
Post-order traversal: 12 18 15 24 30 25 23
Binary Search Tree Operations:
1. Insert Node
2. Display Tree (All Traversals)
3. Exit
Enter your choice: 3
Exiting the program.
```

Lab Program 9

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

Screenshot of Observation:

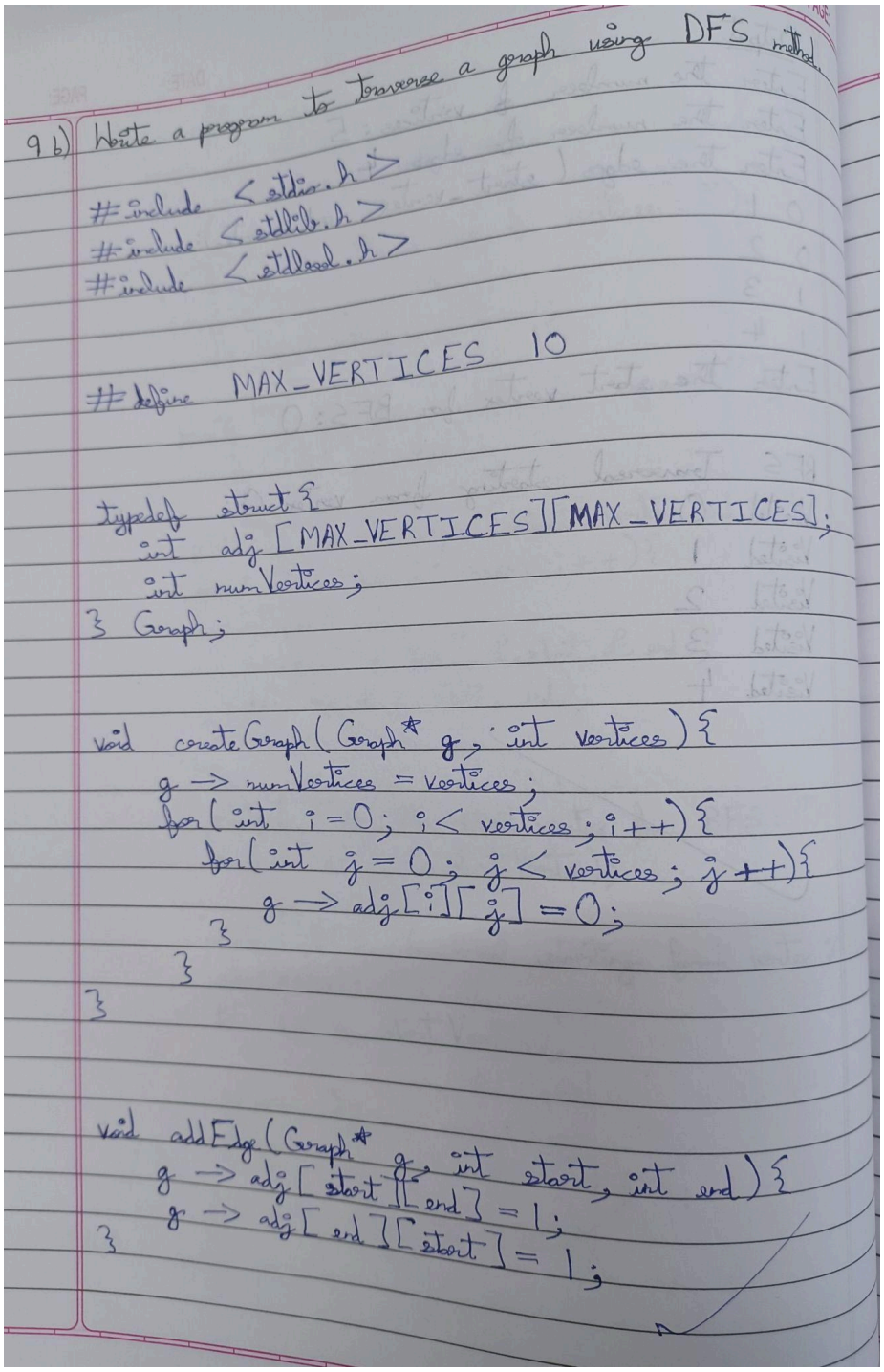


Week -9

DATE: PAGE:

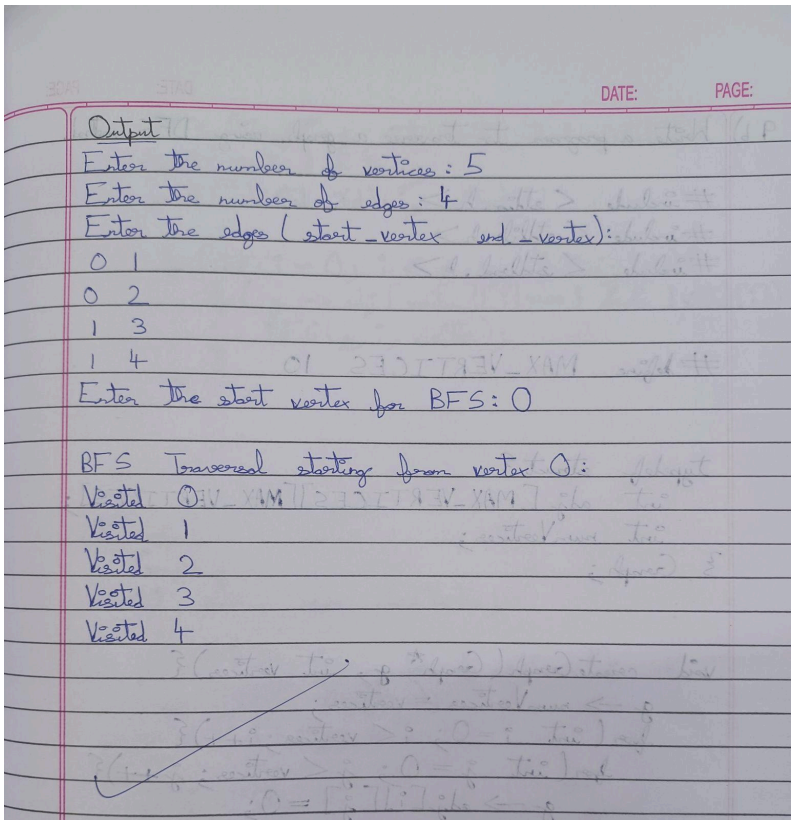
9a) Write a program to traverse a graph using BFS method.

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

#define MAX_VERTICES 10

typedef struct {
    int adj[MAX_VERTICES][MAX_VERTICES];
    int numVertices;
} Graph;

typedef struct {
    int items[MAX_VERTICES];
    int front, rear;
} Queue;

void initQueue(Queue* q) {
    q->front = q->rear = -1;
}

bool isEmpty(Queue* q) {
    return q->front == -1;
}
```



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

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

#define MAX_VERTICES 10

typedef struct {
    int adj[MAX_VERTICES][MAX_VERTICES];
    int numVertices;
} Graph;

void createGraph(Graph* g, int vertices) {
    g->numVertices = vertices;
    for(int i=0; i<vertices; i++) {
        for(int j=0; j<vertices; j++) {
            g->adj[i][j] = 0;
        }
    }
}

void addEdge(Graph* g, int start, int end) {
    g->adj[start][end] = 1;
    g->adj[end][start] = 1;
}
```

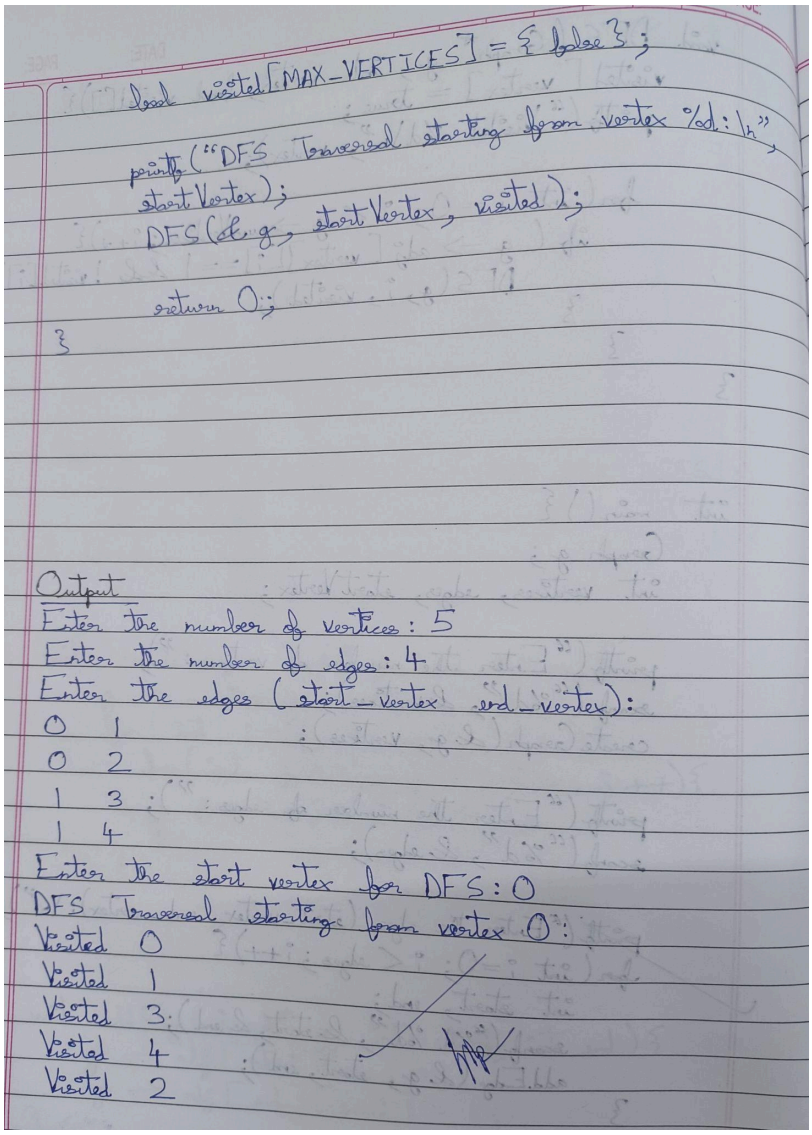


DATE: PAGE:

Output

Enter the number of vertices: 5
Enter the number of edges: 4
Enter the edges (start_vertex end_vertex):
0 1
0 2
1 3
1 4
Enter the start vertex for BFS: 0

BFS Traversal starting from vertex 0:
Visited 0
Visited 1
Visited 2
Visited 3
Visited 4



Code:

9a)

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

#define MAX_VERTICES 10


typedef struct {
    int adj[MAX_VERTICES][MAX_VERTICES];
    int numVertices;
} Graph;


void createGraph(Graph* g, int vertices) {
```

```
    g->numVertices = vertices;
    for (int i = 0; i < vertices; i++) {
        for (int j = 0; j < vertices; j++) {
            g->adj[i][j] = 0;
        }
    }
}


void addEdge(Graph* g, int start, int end) {
    g->adj[start][end] = 1;
    g->adj[end][start] = 1;
}


void BFS(Graph* g, int startVertex) {
    int visited[MAX_VERTICES] = {0};
    int bfsQueue[MAX_VERTICES];
    int front = 0, rear = 0;

    visited[startVertex] = 1;
    bfsQueue[rear++] = startVertex;

    printf("BFS Traversal starting from vertex %d:\n", startVertex);


    while (front < rear) {
        int currentVertex = bfsQueue[front++];
        printf("Visited %d\n", currentVertex);

        for (int i = 0; i < g->numVertices; i++) {
            if (g->adj[currentVertex][i] == 1 && !visited[i]) {
                visited[i] = 1;
                bfsQueue[rear++] = i;
            }
        }
    }
}

int main() {
    Graph g;
    int vertices, edges, startVertex;

    printf("Enter the number of vertices: ");
    scanf("%d", &vertices);
    createGraph(&g, vertices);

    printf("Enter the number of edges: ");
    scanf("%d", &edges);
```

```
    printf("Enter the edges (start_vertex end_vertex):\n");
    for (int i = 0; i < edges; i++) {
        int start, end;
        scanf("%d %d", &start, &end);
        addEdge(&g, start, end);
    }

    printf("Enter the start vertex for BFS: ");
    scanf("%d", &startVertex);

    BFS(&g, startVertex);

    return 0;
}
```

9b)

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

#define MAX_VERTICES 10


typedef struct {
    int adj[MAX_VERTICES][MAX_VERTICES];
    int numVertices;
} Graph;


void createGraph(Graph* g, int vertices) {
    g->numVertices = vertices;
    for (int i = 0; i < vertices; i++) {
        for (int j = 0; j < vertices; j++) {
            g->adj[i][j] = 0;
        }
    }
}


void addEdge(Graph* g, int start, int end) {
    g->adj[start][end] = 1;
    g->adj[end][start] = 1;
}
```

```
void DFS(Graph* g, int vertex, bool visited[]) {
    visited[vertex] = true;
    printf("Visited %d\n", vertex);

    for (int i = 0; i < g->numVertices; i++) {
        if (g->adj[vertex][i] == 1 && !visited[i]) {
            DFS(g, i, visited);
        }
    }
}


int main() {
    Graph g;
    int vertices, edges, startVertex;

    printf("Enter the number of vertices: ");
    scanf("%d", &vertices);
    createGraph(&g, vertices);

    printf("Enter the number of edges: ");
    scanf("%d", &edges);

    printf("Enter the edges (start_vertex end_vertex):\n");
    for (int i = 0; i < edges; i++) {
        int start, end;
        scanf("%d %d", &start, &end);
        addEdge(&g, start, end);
    }


    printf("Enter the start vertex for DFS: ");
    scanf("%d", &startVertex);

    bool visited[MAX_VERTICES] = {false};


    printf("DFS Traversal starting from vertex %d:\n", startVertex);
    DFS(&g, startVertex, visited);

    return 0;
}
```

Output:

9a)

```
Enter the number of vertices: 5
Enter the number of edges: 4
Enter the edges (start_vertex end_vertex):
0 1
0 2
1 3
1 4
Enter the start vertex for BFS: 0
BFS Traversal starting from vertex 0:
Visited 0
Visited 1
Visited 2
Visited 3
Visited 4
```

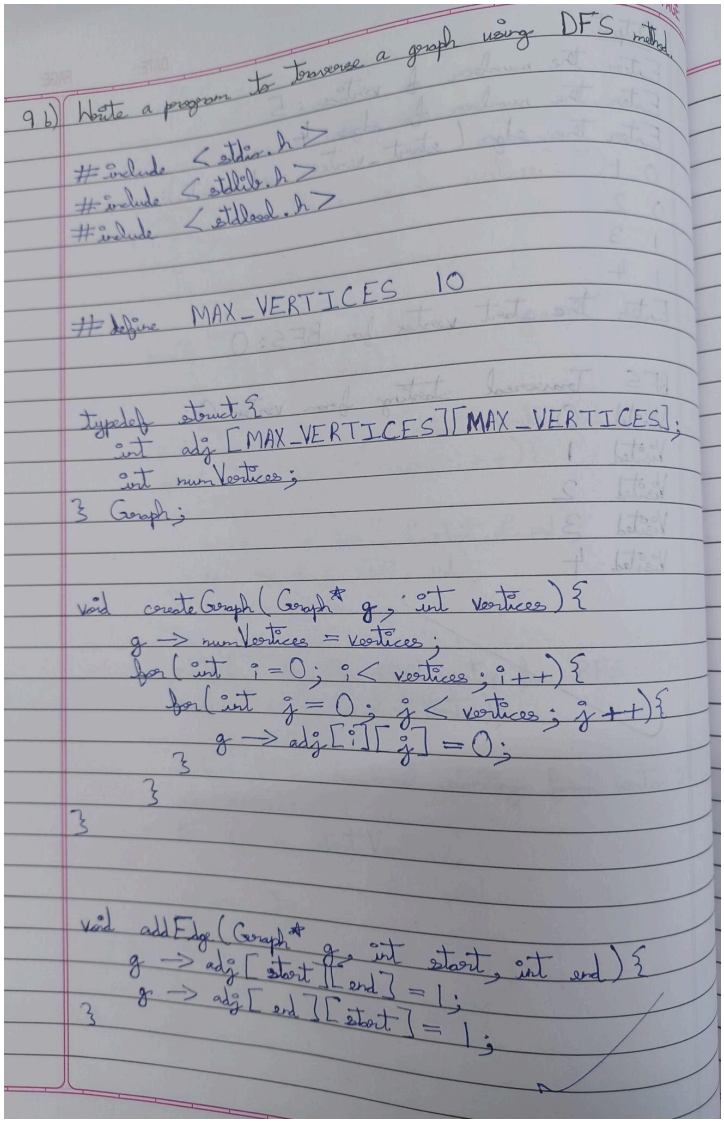
9b)

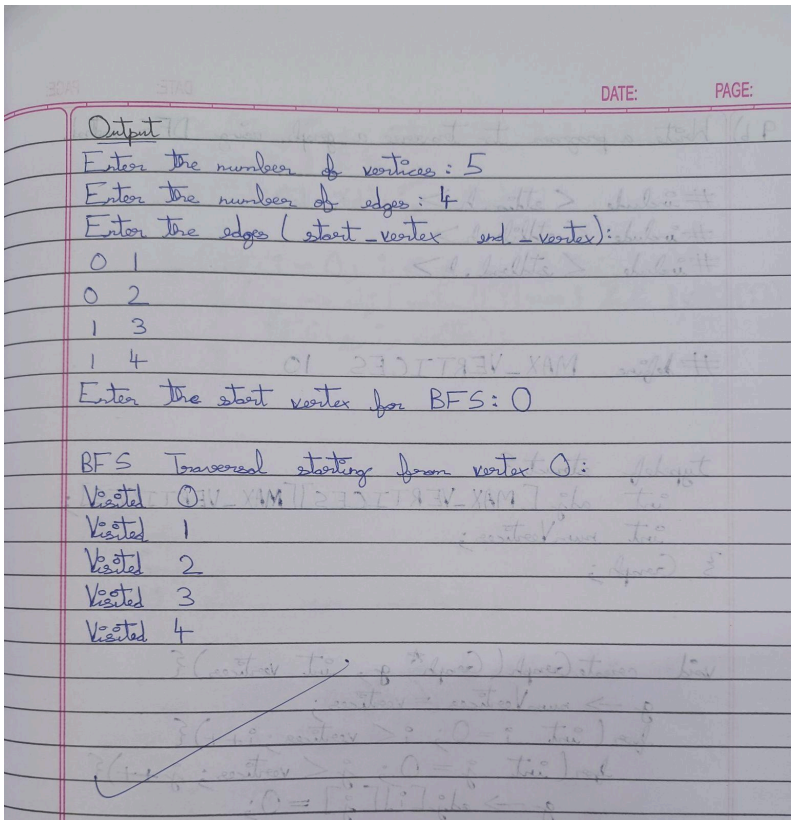
```
Enter the number of vertices: 5
Enter the number of edges: 4
Enter the edges (start_vertex end_vertex):
0 1
0 2
1 3
1 4
Enter the start vertex for DFS: 0
DFS Traversal starting from vertex 0:
Visited 0
Visited 1
Visited 3
Visited 4
Visited 2
```

Lab Program 10

Given a File of N employee records with a set K of Keys (4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are integers. Design and develop a Program in C that uses Hash function H: K -> L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.

Screenshot of Observation:





Code:

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

#define MAX_EMPLOYEES 100


typedef struct {
    int key;
    char name[30];
} Employee;


typedef struct {
    Employee *table[MAX_EMPLOYEES];
    int m;
} HashTable;


void initHashTable(HashTable *ht, int m) {
    ht->m = m;
    for (int i = 0; i < m; i++) {
        ht->table[i] = NULL;
    }
}
```

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


void insert(HashTable *ht, int key, char *name) {
    int index = hashFunction(key, ht->m);
    int originalIndex = index;

    Employee *emp = (Employee*) malloc(sizeof(Employee));
    emp->key = key;
    snprintf(emp->name, sizeof(emp->name), "%s", name);


    while (ht->table[index] != NULL) {

        if (ht->table[index]->key == key) {
            ht->table[index] = emp;
            return;
        }

        index = (index + 1) % ht->m;
        if (index == originalIndex) {
            printf("Hash table is full, cannot insert new record.\n");
            free(emp);
            return;
        }
    }


    ht->table[index] = emp;
    printf("Inserted %s at index %d\n", name, index);
}


Employee* search(HashTable *ht, int key) {
    int index = hashFunction(key, ht->m);
    int originalIndex = index;


    while (ht->table[index] != NULL) {
        if (ht->table[index]->key == key) {
            return ht->table[index];
        }

        index = (index + 1) % ht->m;
        if (index == originalIndex) {
            break;
```

```
        }
    }

    return NULL;
}


void displayHashTable(HashTable *ht) {
    printf("Hash Table contents:\n");
    for (int i = 0; i < ht->m; i++) {
        if (ht->table[i] != NULL) {
            printf("Index %d: Key = %d, Name = %s\n", i, ht->table[i]->key, ht->table[i]->name);
        } else {
            printf("Index %d: Empty\n", i);
        }
    }
}

int main() {
    HashTable ht;
    int m;

    printf("Enter the size of the hash table (m): ");
    scanf("%d", &m);

    initHashTable(&ht, m);

    int choice, key;
    char name[30];

    while (1) {
        printf("\nMenu:\n");
        printf("1. Insert Employee Record\n");
        printf("2. Search Employee by Key\n");
        printf("3. Display Hash Table\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter employee key (4-digit): ");
                scanf("%d", &key);
                printf("Enter employee name: ");
                scanf("%s", name);
                insert(&ht, key, name);
                break;
            case 2:
                printf("Enter employee key to search: ");
```

```
            scanf("%d", &key);
            Employee *emp = search(&ht, key);
            if (emp != NULL) {
                printf("Employee found: Key = %d, Name = %s\n", emp->key, emp->name);
            } else {
                printf("Employee with key %d not found.\n", key);
            }
            break;
        case 3:
            displayHashTable(&ht);
            break;
        case 4:
            printf("Exiting program.\n");
            return 0;
        default:
            printf("Invalid choice. Try again.\n");
    }
  }

  return 0;
}
```

Output:

```
Enter the size of the hash table (m): 5

Menu:
1. Insert Employee Record
2. Search Employee by Key
3. Display Hash Table
4. Exit
Enter your choice: 1
Enter employee key (4-digit): 1234
Enter employee name: John
Inserted John at index 4

Menu:
1. Insert Employee Record
2. Search Employee by Key
3. Display Hash Table
4. Exit
Enter your choice: 1
Enter employee key (4-digit): 5678
Enter employee name: Alice
Inserted Alice at index 3
```

```
Menu:
1. Insert Employee Record
2. Search Employee by Key
3. Display Hash Table
4. Exit
Enter your choice: 3
Hash Table contents:
Index 0: Empty
Index 1: Empty
Index 2: Empty
Index 3: Key = 5678, Name = Alice
Index 4: Key = 1234, Name = John

Menu:
1. Insert Employee Record
2. Search Employee by Key
3. Display Hash Table
4. Exit
Enter your choice: 2
Enter employee key to search: 1234
Employee found: Key = 1234, Name = John
```

```
Menu:
1. Insert Employee Record
2. Search Employee by Key
3. Display Hash Table
4. Exit
Enter your choice: 4
Exiting program.
```