VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT

on

DATA STRUCTURES IN C

Submitted by

SRUJAN K R (1BM23CS340)

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-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 "**OBJECT ORIENTED JAVA PROGRAMMING**" carried out by **SRUJAN K R(1BM23CS340),** who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2024-25. The Lab report has been approved as it satisfies the academic requirements in respect of **Object-Oriented Java Programming Lab - (23CS3PCOOJ)** work prescribed for the said degree.

Geeta N

Associate Professor,
Department of CSE,
BMSCE, Bengaluru

Dr. Kavitha Sooda

Professor and Head,
Department of CSE
BMSCE, Bengaluru

INDEX

https://github.com/Srujan4812/1BM23CS340-DS-C

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

Observation:

```
(1) #include stdio.h

Struct Stack {
    int top;
    int items [max];
};

Void intstack (Strucct Stack *s) {
    S->top = -1;
}

int is-full (Strucct stack *s) {
    return s->top == max-1;
}

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

Void push (Struct stack *s, int item) {
    if (! isfull (s)) {
    S->items [++s -> top] = item;
    Printf (" pushed %d onto stack \n");
    }
    else {
        Print (" stack overflow );
    }
}
```

```
int
void pop (Struct Stack * s) {
    if (!isempty(s)) {
        return s->items [s->top --];
    } else {
        printf ("Stack underflow);
        return -1;
    }
}

int main () {
 Struct Stack * s = (Struct Stack*) malloc (
                       size of (Struct Stack));

push(2s, 10);
push( 2s, 20);
pop (2s, 30);
pop(2s, 30);
peek(2s);

}
```

Output :

Stack size = 3

Step		
1)	40 30 20 10	push(10)
2)		push(20)
3)		push(30)
4)		push(40)

"Stack overflow"

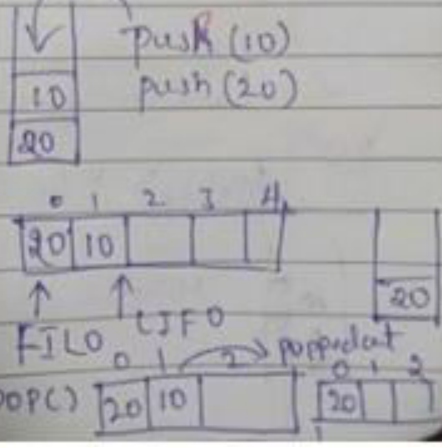
Step	
1)	pop()
2)	pop()
3)	pop()
4)	pop()

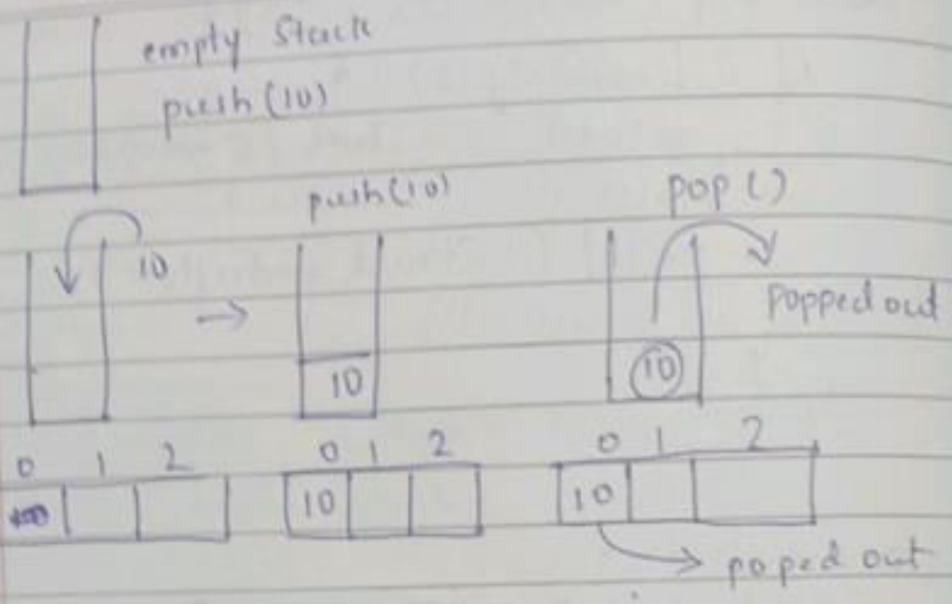
" Stack underflow "

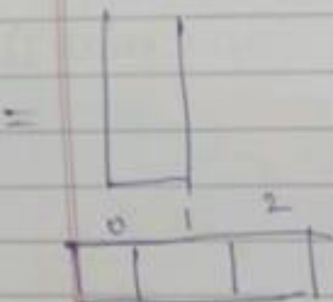
Peek (sp) :

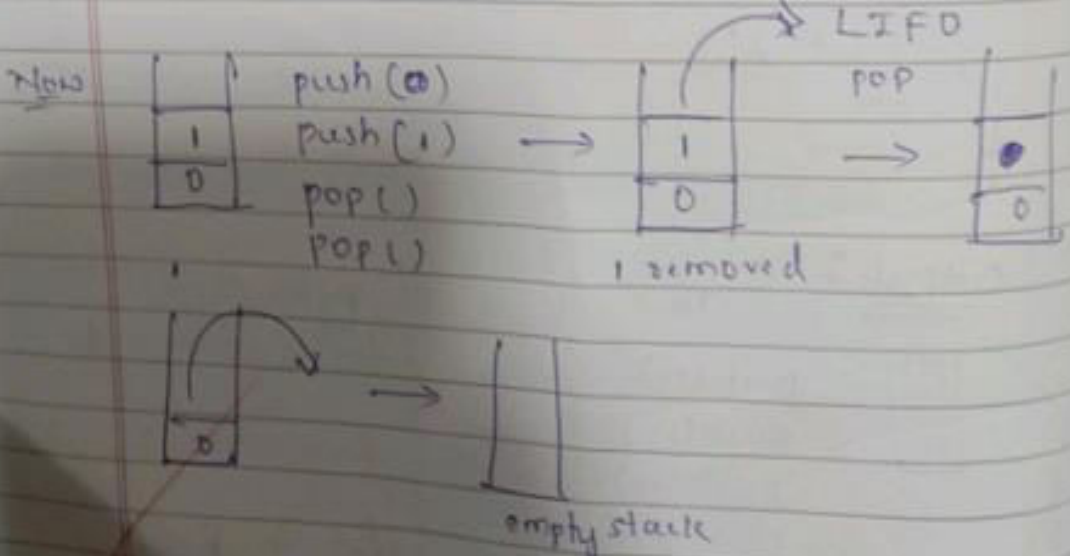
top of stack = 10

free Stack();









LIFO Last in first out

Code:

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

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

void push(Stack *s, int value) {
    if (s->top == MAX - 1) {
        printf("Stack Overflow\n");
        return;
    }
    s->arr[++(s->top)] = value;
    printf("Pushed %d\n", value);
}

void pop(Stack *s) {
    if (s->top == -1) {
        printf("Stack Underflow\n");
        return;
    }
    printf("Popped %d\n", s->arr[(s->top)--]);
}

void display(Stack *s) {
    if (s->top == -1) {
        printf("Stack is Empty\n");
```

```
        return;
    }
    printf("Stack elements: ");
    for (int i = 0; i <= s->top; i++)
        printf("%d ", s->arr[i]);
    printf("\n");
}

int main() {
    Stack s;
    s.top = -1;
    int choice, value;

    while (1) {
        printf("\nChoose an option:\n");
        printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push(&s, value);
                break;
            case 2:
                pop(&s);
                break;
            case 3:
```

```
            display(&s);
            break;
        case 4:
            return 0;
        default:
            printf("Invalid choice!\n");
      }
   }
}
```

Output:

```
srujan  R@SRUJAN MINGW64 ~
$ Choose an option:
1. Push
2. Pop
$ Choose an option:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 10
Pushed 10

Choose an option:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 20
Pushed 20

Choose an option:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack elements: 10 20

Choose an option:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Popped 20

Choose an option:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack elements: 10

Choose an option:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 4
```

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)

Observation:

(2) Prefix to postfix

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

int prec(char c) {
 if (c == '^') return 3;
 if (c == '/' || c == '*') return 2;
 if (c == '+' || c == '-') return ;

 return -1; }

char associativity (char c) {
        if (c == '^') return 'R' ;
        return 'L' ;
   }

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

 for (int i = 0 ; i < len ; i++ ) {
     char c = s[i] ;
```

```
if ((c>='a' && c<='z') || (c>='A' && c<='Z') ||
   (c>='0' && c<='9')) {
     result [result Index + 1]= c ;
} else if (c== '(') {
      stack [++ stack Index]= c ;
   }
   else if (c=='')')
   {
 While (stackIndex >=0 && stack [stackIndex]
                                  != '(')
 {
result [result Index ++ ] = stack [stack Index--];
 }
 stack Index -- ; } else {
While(stackIndex>=0 && (prec(c) < prec(stack[
   stackIndex])) || (prec(c) == prec (stack [stackIndex]
   == '('))) {
     result [result Index++] = stack[stackIndex--]
  }
 stack [++stack Index]= c ;
 }
 }
 While (stackIndex >=0 ) {
 result [resultIndex++ ] = stack [stackIndex--];
 }
result [result Index ]= '\0' ;
printf (" %s \n", result);
free (result);
free (stack) ; }
```

```
int main () {
Char exp[] = "a+b*((^d-e)^(f+g*h)-i");
    I+p (exp);
    return 0;
}
```

output

I/p a+b*((^d-e)^(f+g*h)-i

output I/p (A+B)*(C-D)

postfix exp: (AB+)*(CD-)

expected o/p AB+CD-*

10/10

my o/p
from execution :

(A+B)*(C-D)

AB+CD-*

8/10

Code:

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

#define MAX 100

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

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

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

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

void infixToPostfix(char* infix, char* postfix) {
   int i = 0, j = 0;
   char c, temp;

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

```
        if (isalnum(c)) {
            postfix[j++] = c;  // Append operand
        } else if (c == '(') {
            push(c);
        } else if (c == ')') {
            while (top != -1 && (temp = pop()) != '(') {
                postfix[j++] = temp;  // Pop till '('
            }
        } else {  // Operator
            while (top != -1 && precedence(stack[top]) >= precedence(c)) {
                postfix[j++] = pop();
            }
            push(c);
        }
    }

    while (top != -1) {
        postfix[j++] = pop();
    }
    postfix[j] = '\0';
}

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

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

    infixToPostfix(infix, postfix);
```

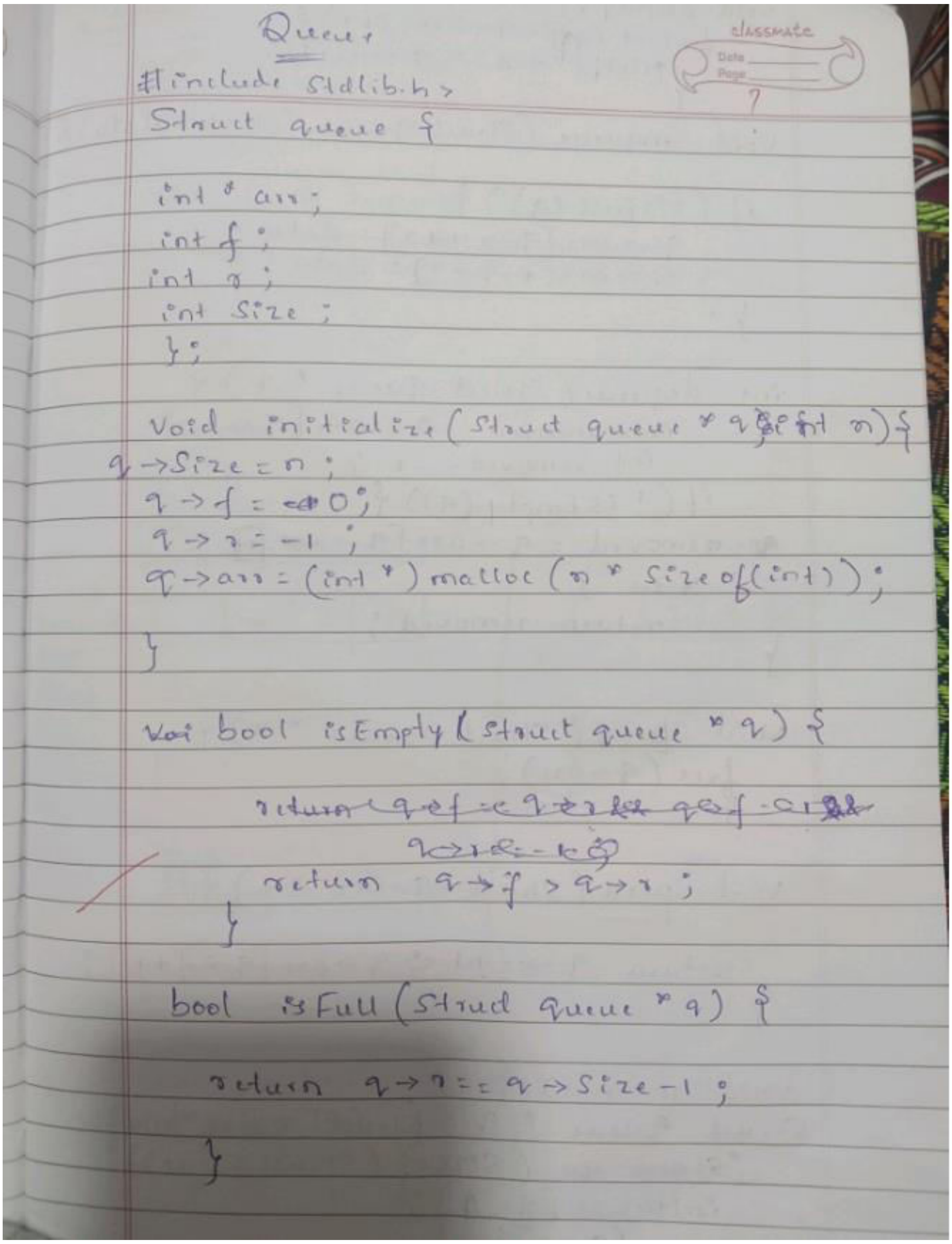
```
    printf("Postfix Expression: %s\n", postfix);


    return 0;
}
```

Output:

```
Enter a valid parenthesized infix expression: (A+B)*(c-D)
Postfix expression: AB+cD-*
```

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

Observation:



```
Queue
#include stdlib.h>
Struct queue {

int * arr;
int f;
int r;
int size;
};

Void initialize(Struct queue * q, int n){
q->Size = n;
q->f = 0;
q->r = -1;
q->arr = (int *) malloc(n * Sizeof(int));

}

bool isEmpty(Struct queue * q) {

return q->f > q->r;
}

bool isFull(Struct Queue * q) {

return q->r == q->Size - 1;

}
```

```
void displayq(...
  for(int i=0; i< q->size; i++){
    printf("%d ", q->arr[i]);
  }

void enqueue (struct queue *q, int data)

  if (!isFull(q)) {
    ~~q->arr[q->r++] = data;~~
    q->arr[++q->r] = data;
  }
}

int dequeue(struct queue * q) {
    ~~int removed = q->arr[q->f~~
    int removed = -1 ;
  if(! isEmpty(q)) {
  removed = q->arr [q-> f++] ;
  }
    return removed;
}

void freeq( struct queue * q) {
  free (q->arr);
  }

void peek( struct queue * q) {

  return q->arr [q->f+1];
}

void main () {
struct queue * q = (struct queue *)malloc
  (sizeof(struct queue));
  initalize (q, 4);
enqueue (q, 10);
dequeue(q);
dequeue (q);
enqueue (q5);
enqueue (q, 6);
} displayq(q);
```

O/P

Size = 4

enqued 4 elements 1 2 3 4

1	2	3	4

added from rear

enqueue again
enqueue(5);
Queue overflow

dequeue();

1	2	3	4

removed from front

	2	3	4

dequeue();

		3	4

dequeue();

			4

dequeue();

FIFO

0	1	2
4	2	

4

0	1	2
	2	4

O/P
5

5 10

Queue underflow
enqueue(1); enqueue(2); enqueue(3);
display(q);

1 2 3 4

Peek(q);
1

Executed
14/10

Output

1) Enqueue
2) Dequeue
3) Display Queue
4) Exit

1)
2 4 5 6

2) 2 4 5 6

3) Dequeued 2

4) == Code execution Successful ==

3) Dequeued 4

3) Dequeued 5

3) Dequeued 6

3) Queue underflow

4) == Code execution successful ==

Code:

```
#include <stdio.h>
#define MAX 5

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

void insert(int val) {
    if (rear == MAX - 1) {
        printf("Queue Overflow\n");
    } else {
        if (front == -1) front = 0;
        queue[++rear] = val;
        printf("Inserted %d\n", val);
    }
}

void delete() {
    if (front == -1 || front > rear) {
        printf("Queue Underflow\n");
    } else {
        printf("Deleted %d\n", queue[front++]);
        if (front > rear) front = rear = -1;
    }
}

void display() {
    if (front == -1) {
        printf("Queue is Empty\n");
```

```
    } else {
        printf("Queue elements: ");
        for (int i = front; i <= rear; i++) printf("%d ", queue[i]);
        printf("\n");
    }
}

int main() {
    int choice, value;
    while (1) {
        printf("\nLinear Queue Operations:\n1. Insert\n2. Delete\n3. Display\n4. Exit\nEnter
your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insert(value);
                break;
            case 2:
                delete();
                break;
            case 3:
                display();
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice!\n");
        }
```

```
    }
}
```

Output:

```
srujan  R@SRUJAN MINGW64 ~
$ Choose an option:
1. Push
2. Pop
$ Choose an option:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 10
Pushed 10

Choose an option:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 20
Pushed 20

Choose an option:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack elements: 10 20

Choose an option:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Popped 20

Choose an option:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack elements: 10

Choose an option:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 4
```

3. b) 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

Observation:

Circular queues

1) Insertion, Deletion, display

```
struct Cqueue Q {
    int * arr;
    int size;
    int f;
    int r;
};

void initialize (struct Q * q, int n) {
    q->arr = (int *) malloc (n * sizeof(int));
    q->size = n;
    q->f = -1;
    q->r = -1;
}

bool isEmpty ( struct Q * q) {
    return q->f == -1 && q->r == -1 ;
}

bool isFull ( struct Q * q ) {
    return q->(r+1) % (q->size) == q->f ;
}

void insertion ( struct Q * q, int data) {
    if (!isFull(q)) {
        q->f = 0; // make f = 0; }
        q->r = q->(r+1) % (q->size) ;
        q->arr[q->r] = data;
    } else {
    printf(" Overflow ");
    }
}
```

```
~~void~~ int deletion(struct Q * q) {
    ~~if (q->size==~~
    if (!isEmpty(q)) {
        q->f = (q->f+1) % (q->size);
        int ele = q->arr[q->f];
        return ele;
    } else {
    return -1;
    }
}

void display(struct Q * q) {

for (int i=q->f ; i!=q->r ; i=(i+1)% (q->size
    printf("%d\n", q->arr[i]);
    }

void main() {
struct Q * q =(struct Q *) malloc( sizeof(struct Q
 initialize(q, 5);
 insertion(q, 6);
 insertion (q, 8);
 deletion (q);
 display(q);

}
```

Switch case

~~insertion~~ if

Output

1) Insert
2) Delete
3. Display
4. Exit

1 ↵
Enter value to insert : 5 ↵
1 ↵
Enter value to insert : 4 ↵
1 ↵
Enter value to insert : 8 ↵
1 ↵
Enter value to insert : 4 ↵

"Queue Overflow"
3. Queue elements 5 4 8 4
2. Deleted value 5
2. Deleted value 4
2. Deleted value 8.
2 Deleted value 4
2. "Queue underflow"

Executed

Code:

```
#include <stdio.h>
#define MAX 5

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

void insert(int val) {
   if ((front == 0 && rear == MAX - 1) || (front == rear + 1)) {
      printf("Queue Overflow\n");
   } else {
      if (front == -1) {
         front = rear = 0;
      } else if (rear == MAX - 1) {
         rear = 0;
      } else {
         rear++;
      }
      queue[rear] = val;
      printf("Inserted %d\n", val);
   }
}

void delete() {
   if (front == -1) {
      printf("Queue Underflow\n");
   } else {
```

```
        printf("Deleted %d\n", queue[front]);
        if (front == rear) {
            front = rear = -1;
        } else if (front == MAX - 1) {
            front = 0;
        } else {
            front++;
        }
    }
}

void display() {
    if (front == -1) {
        printf("Queue is Empty\n");
    } else {
        printf("Queue elements: ");
        if (rear >= front) {
            for (int i = front; i <= rear; i++) printf("%d ", queue[i]);
        } else {
            for (int i = front; i < MAX; i++) printf("%d ", queue[i]);
            for (int i = 0; i <= rear; i++) printf("%d ", queue[i]);
        }
        printf("\n");
    }
}

int main() {
    int choice, value;
    while (1) {
```

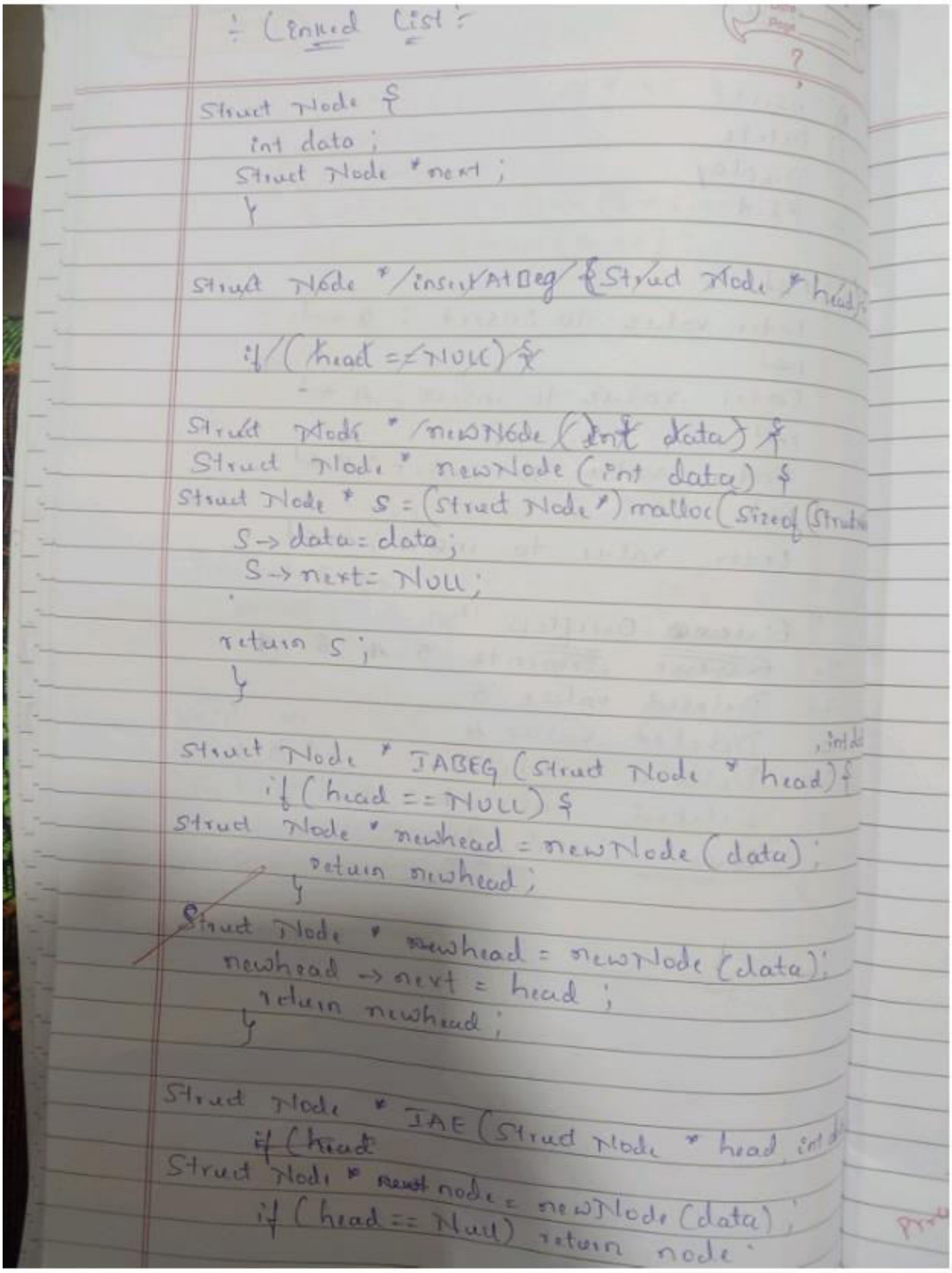
```
        printf("\nCircular Queue Operations:\n1. Insert\n2. Delete\n3. Display\n4. Exit\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insert(value);
                break;
            case 2:
                delete();
                break;
            case 3:
                display();
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice!\n");
        }
    }
}
```

Output:

```
srujan  R@SRUJAN MINGW64 ~
$ Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 10
Inserted 10

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

Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 30
Inserted 30

Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue elements: 10 20 30

Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted 10

Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue elements: 20 30

Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 40
Inserted 40
```

4. WAP 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. Display the contents of the linked list.

Observation:



÷ Linked List ÷

```
Struct Node {
  int data ;
  Struct Node *next ;
}

Struct Node */insertAtBeg/ (Struct Node * head)

if/(head =/ NULL) {

Struct Node * /newNode (int data) {
Struct Node * newNode (int data) {
Struct Node * s = (Struct Node *) malloc( sizeof (Struct
  s -> data = data ;
  s -> next = NULL ;

  return s ;
}

Struct Node * IABEG (Struct Node * head, int data) {
  if (head == NULL) {
Struct Node * newhead = newNode (data) ;
  return newhead ;
  }
Struct Node * newhead = newNode (data) ;
newhead -> next = head ;
  return newhead ;
}

Struct Node * IAE (Struct Node * head, int d
  if (head
Struct Node * node = newNode (data) ;
  if (head == Null) return node ;
```

Linked List

Insert

Accessing

While
```
Struct Node *s = head;
  While (s->next != NULL) {
        s = s->next;
    }
    s->next = node;
    return head;
}

Void display (Struct Node * head) {
  Struct Node *s
  if (head == NULL) {
      printf (" Empty ");
  } else {
  While (s->next != NULL) {
     printf (" %d ", s->data);
         s = s->next;
     }
     printf (" \n ");
}

Void main() {
Struct Node * head 0 = newNode(5);
Struct Node * first 0 = newNode(6);
Struct Node * Second 0 = newNode(7);

head->next = first;
first->next = Second;

IABEG (head, 6);
IAE (head, 12);

display(head);

}
```

O/P

Enter your choice: 1
Choose an option
1) Insert at Beginning
2) Insert at End
3) Print List
4) Exit

1) Enter data to insert at beginning: 1
1) Enter data to insert at beginning: 2
1) Enter data to insert at beginning: 3
1) Enter data to insert at beginning 4

2) Enter data 5
2) Enter data 6

3) 4 -> 3 -> 2 -> 1 -> 5 -> 6 -> NULL

4) Exitting...

—— || Code execution Successful || ——

Code:

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

// Node structure for Singly Linked List
struct Node {
    int data;
    struct Node* next;
};

struct Node* head = NULL;

// Create a linked list
void createLinkedList(int n) {
    struct Node *newNode, *temp;
    int data, i;

    for (i = 0; i < n; i++) {
        printf("Enter data for node %d: ", i + 1);
        scanf("%d", &data);

        newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->data = data;
        newNode->next = NULL;

        if (head == NULL) {
            head = newNode;
        } else {
            temp = head;
```

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

// Insert at the beginning
void insertAtBeginning(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = head;
    head = newNode;
    printf("Node inserted at the beginning.\n");
}

// Insert at a specific position
void insertAtPosition(int data, int position) {
    struct Node *newNode, *temp;
    int i;

    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;

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

```
        temp = head;
        for (i = 1; i < position - 1 && temp != NULL; i++) {
            temp = temp->next;
        }

        if (temp != NULL) {
            newNode->next = temp->next;
            temp->next = newNode;
            printf("Node inserted at position %d.\n", position);
        } else {
            printf("Position out of range.\n");
        }
    }
}

// Insert at the end
void insertAtEnd(int data) {
    struct Node *newNode, *temp;

    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;

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

```
        }
        temp->next = newNode;
    }
    printf("Node inserted at the end.\n");
}

// Display the linked list
void displayList() {
    struct Node* temp = head;

    if (head == NULL) {
        printf("The list is empty.\n");
    } else {
        printf("Linked list contents: ");
        while (temp != NULL) {
            printf("%d -> ", temp->data);
            temp = temp->next;
        }
        printf("NULL\n");
    }
}

int main() {
    int choice, data, position, n;

    while (1) {
        printf("\nSingly Linked List Operations:\n");
        printf("1. Create Linked List\n2. Insert at Beginning\n3. Insert at Position\n4. Insert at
End\n5. Display List\n6. Exit\n");
        printf("Enter your choice: ");
```

```
scanf("%d", &choice);

switch (choice) {
  case 1:
    printf("Enter the number of nodes: ");
    scanf("%d", &n);
    createLinkedList(n);
    break;
  case 2:
    printf("Enter data to insert at beginning: ");
    scanf("%d", &data);
    insertAtBeginning(data);
    break;
  case 3:
    printf("Enter data to insert: ");
    scanf("%d", &data);
    printf("Enter position: ");
    scanf("%d", &position);
    insertAtPosition(data, position);
    break;
  case 4:
    printf("Enter data to insert at end: ");
    scanf("%d", &data);
    insertAtEnd(data);
    break;
  case 5:
    displayList();
    break;
  case 6:
```

```
            return 0;

          default:

            printf("Invalid choice!\n");

        }

    }

}
```

Output:

```
$ Singly Linked List Operations:
1.Singly Linked List Operations:
1. Create Linked Listg
2. Insert at Beginning
3. Insert at Position
4. Insert at End
5. Display List
6. Exitour choice: 1
Enter your choice: 1nodes: 3
Enter the number of nodes: 3
Enter data for node 1: 10
Enter data for node 2: 20
Enter data for node 3: 30
Singly Linked List Operations:
Singly Linked List Operations:
1. Create Linked Listg
2. Insert at Beginning
3. Insert at Position
4. Insert at End
5. Display List
6. Exitour choice: 2
Enter your choice: 2 at beginning: 5
Enter data to insert at beginning: 5
Node inserted at the beginning.
Singly Linked List Operations:
Singly Linked List Operations:
1. Create Linked Listg
2. Insert at Beginning
3. Insert at Position
4. Insert at End
5. Display List
6. Exitour choice: 3
Enter your choice: 3: 25
Enter data to insert: 25
Enter position: 3position 3.
Node inserted at position 3.
Singly Linked List Operations:
Singly Linked List Operations:
1. Create Linked Listg
2. Insert at Beginning
3. Insert at Position
4. Insert at End
5. Display List
6. Exitour choice: 4
Enter your choice: 4 at end: 40
Enter data to insert at end: 40
Node inserted at the end.
Singly Linked List Operations:
Singly Linked List Operations:
1. Create Linked Listg
2. Insert at Beginning
3. Insert at Position
4. Insert at End
5. Display List
6. Exitour choice: 5
Enter your choice: 5: 5 -> 10 -> 20 -> 25 -> 30 -> 40 -> NULL
Linked list contents: 5 -> 10 -> 20 -> 25 -> 30 -> 40 -> NULL
Singly Linked List Operations:
Singly Linked List Operations:
1. Create Linked Listg
2. Insert at Beginning
3. Insert at Position
4. Insert at End
5. Display List
6. Exitour choice: 6
```

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

Observation:

11/11/24

WAP to Implement LL
i) Create a LL
ii) Deletion of 1st ele, Specified ele & last ele
c) Display

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
Struct Node {
  int data;
  Struct Node * next;
  };

bool
Voi Struct Node * newnode (int data);
Struct Node * S = (Struct Node *)malloc (Size of
          Struct Node ));
  S -> data = data;
  S -> next = null;
  return S;
  }

bool isEmpty (Struct Node * head) {
   return head == null;
  }

~~bool isFull (Struct Node * head) {~~

Struct Node * deletefirst (Struct Node * head){
  if (!isEmpty(head)) {
        ~~head = head~~
  Struct Node * ptr = head;
        head = head -> next;
        free(ptr);
        return head; }
```

```
Struct Node* dlast(Struct Node* head) {
 Struct Node* prev = head;
 Struct Node* ptr = head->next;

  if (! is Empty (head)) {
     While( ptr->next != null) {
          ptr = ptr -> next;
          head = prev -> next;
          prev -> next = null;
          free(ptr);
          return head;
     }
  }
```

```
Struct Node * dspecificdele (Struct Node * head,
                               int ele) {
Struct node * ptr= head ;
   While (ptr -> data = = ele) {
        ptr = ptr -> next ;
     }
   While(
   Struct Node * prev = head ;
   While ( ptr -> next = = ptr) {
           prev = prev -> next ;
     }

  prev -> next = ptr -> next ;
   free(ptr);
   return head ;
}

void main () {
   insert (2);
   insert (3);
   insert (4);

 deletedele (4, head);
 delLast (head);
 deletef (head);

}
```

O/P

1 → 2 → 3 → 4 → 5

1) Delete first element
   2 → 3 → 4 → 5

2) Delete specified ele 4
   2 → 3 → 5

3) Delete last ele

Code:

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

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

struct Node* head = NULL;

void createLinkedList(int n) {
    struct Node *newNode, *temp;
    int data, i;

    for (i = 0; i < n; i++) {
        printf("Enter data for node %d: ", i + 1);
        scanf("%d", &data);

        newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->data = data;
        newNode->next = NULL;

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

```
            }
            temp->next = newNode;
        }
    }
}

void deleteFirst() {
    if (head == NULL) {
        printf("The list is empty. No node to delete.\n");
        return;
    }
    struct Node* temp = head;
    head = head->next;
    free(temp);
    printf("First node deleted.\n");
}

void deleteSpecified(int key) {
    if (head == NULL) {
        printf("The list is empty. No node to delete.\n");
        return;
    }

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

    if (temp != NULL && temp->data == key) {
        head = temp->next;
        free(temp);
        printf("Node with data %d deleted.\n", key);
```

```
        return;
    }

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

    if (temp == NULL) {
        printf("Node with data %d not found.\n", key);
        return;
    }

    prev->next = temp->next;
    free(temp);
    printf("Node with data %d deleted.\n", key);
}

void deleteLast() {
    if (head == NULL) {
        printf("The list is empty. No node to delete.\n");
        return;
    }

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

    if (temp->next == NULL) {
        head = NULL;
        free(temp);
```

```
        printf("Last node deleted.\n");
        return;
    }

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

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

void displayList() {
    struct Node* temp = head;

    if (head == NULL) {
        printf("The list is empty.\n");
    } else {
        printf("Linked list contents: ");
        while (temp != NULL) {
            printf("%d -> ", temp->data);
            temp = temp->next;
        }
        printf("NULL\n");
    }
}
```

```
int main() {
    int choice, data, n;

    while (1) {
        printf("\nSingly Linked List Operations:\n");
        printf("1. Create Linked List\n2. Delete First\n3. Delete Specified\n4. Delete Last\n5.
Display List\n6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the number of nodes: ");
                scanf("%d", &n);
                createLinkedList(n);
                break;
            case 2:
                deleteFirst();
                break;
            case 3:
                printf("Enter the value to delete: ");
                scanf("%d", &data);
                deleteSpecified(data);
                break;
            case 4:
                deleteLast();
                break;
            case 5:
                displayList();
                break;
```

```
            case 6:
                return 0;
            default:
                printf("Invalid choice!\n");
        }
    }
}
```

Output:

```
srujan  R@SRUJAN MINGW64 ~
$ Singly Linked List Operations:
1. Create Linked List
2. Delete First
3. Delete Specified
4. Delete Last
5. Display List
6. Exit
Enter your choice: 1
Enter the number of nodes: 3
Enter data for node 1: 10
Enter data for node 2: 20
Enter data for node 3: 30

Singly Linked List Operations:
1. Create Linked List
2. Delete First
3. Delete Specified
4. Delete Last
5. Display List
6. Exit
Enter your choice: 5
Linked list contents: 10 -> 20 -> 30 -> NULL

Singly Linked List Operations:
1. Create Linked List
2. Delete First
3. Delete Specified
4. Delete Last
5. Display List
6. Exit
Enter your choice: 2
First node deleted.

Singly Linked List Operations:
1. Create Linked List
2. Delete First
3. Delete Specified
4. Delete Last
5. Display List
6. Exit
Enter your choice: 5
Linked list contents: 20 -> 30 -> NULL

Singly Linked List Operations:
1. Create Linked List
2. Delete First
3. Delete Specified
4. Delete Last
5. Display List
6. Exit
Enter your choice: 3
Enter the value to delete: 30
Node with data 30 deleted.

Singly Linked List Operations:
1. Create Linked List
2. Delete First
3. Delete Specified
4. Delete Last
5. Display List
6. Exit
Enter your choice: 5
Linked list contents: 20 -> NULL
```

6. WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists, Implement Single Link List to simulate Stack & Queue Operations.

Observation:

Lab - 7

W.A.p to implement LL with following operations.

(i) Sort (ii) Reverse (iii) Concatenate.

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

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

struct node * Createlinkedlist () {
    struct node * head = Null;
    struct node * temp = Null;
    struct node * newNode;
    int n, value;
    printf("Number of elements");
    scanf("%d", &n);
    for (int i=0; i<n; i++) {
        printf("Enter");
        scanf("%d", &value);

    newNode = (struct node *) malloc(sizeof
                (struct node));
    newnode -> data = Value;
    newNode -> next = Null;

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

```
return head ;
}

void SortLinkedList (Struct Node* head) {
    if ( *head = = Null) return ;
  Struct Node * Currenthead = *head ;
  Struct Node * index = Null ;
  int temp ;

While (current != Null) {
    index = Current -> null ;
  While (index != Null) {
      if (Current -> data > index -> data) {
    temp = Current -> data ;
    Current -> data = index -> data ;
    index -> data = data temp ;
    }
   index = index -> next ;
   }
   Current = Current -> next ;
 }
}

Void reverselinkedList ( Struct Node * head) {
 Struct node * prev = Null ;
 * Current = * head ;
 * next = Null ;
  While (Current != Null) {
    next = Current -> next ;
    Current -> next = prev ;
    prev = Current ;
    Current = next ;
  }
```

```
struct node * Concatenatelist (struct node
 head1, struct node * head) {
  if (head != NULL) return head2;
   if (head2 == NULL) return head1;

  struct node * temp = head1;
  While (temp->next != NULL) {
        temp = temp -> next ; }
  temp ->next = head2;
   return head1 ;
 }

Void main() {

  // switch case
    CreateLinkedList() ;
    SortLinkedList() ;
 }  reverse linked list() ;
    Concatenate_list() ;
    }
```

O/P

2/12

O/P

(i) Enter values for list1 : 1 2 3 4 5 6

(ii) Enter values for list2 : 1 2 3 4 5 6

(iii) Sort reverse list1 : 6 5 4 3 2 1

IV Concatenate 1 & 2 1 2 3 4 5 6 6 5 4 3 2 1

V Sortlist : 1 1 2 2 3 3 4 4 5 5 6 6

V Displaylist :

1 1 2 2 3 3 4 4 5 5 6 6

Code:

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

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

struct Node* head = NULL;

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

void sortList() {
    struct Node* i;
```

```
    struct Node* 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) {
                temp = i->data;
                i->data = j->data;
                j->data = temp;
            }
        }
    }
    printf("Linked list sorted.\n");
}

void reverseList() {
    struct Node *prev = NULL, *current = head, *next = NULL;
    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    head = prev;
    printf("Linked list reversed.\n");
}

void concatenateLists(struct Node* head1, struct Node* head2) {
    if (head1 == NULL) {
        head = head2;
```

```
        return;
    }
    struct Node* temp = head1;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = head2;
    head = head1;
    printf("Linked lists concatenated.\n");
}

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

int main() {
    int choice, data, n, i;
    struct Node *list1 = NULL, *list2 = NULL;
```

```
    while (1) {
        printf("\nSingly Linked List Operations:\n");
        printf("1. Insert at End\n2. Sort List\n3. Reverse List\n4. Concatenate Two Lists\n5.
Display List\n6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to insert at end: ");
                scanf("%d", &data);
                insertAtEnd(data);
                break;
            case 2:
                sortList();
                break;
            case 3:
                reverseList();
                break;
            case 4:
                printf("Enter number of nodes for first list: ");
                scanf("%d", &n);
                for (i = 0; i < n; i++) {
                    printf("Enter data for node %d of list1: ", i + 1);
                    scanf("%d", &data);
                    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
                    newNode->data = data;
                    newNode->next = list1;
                    list1 = newNode;
                }
```

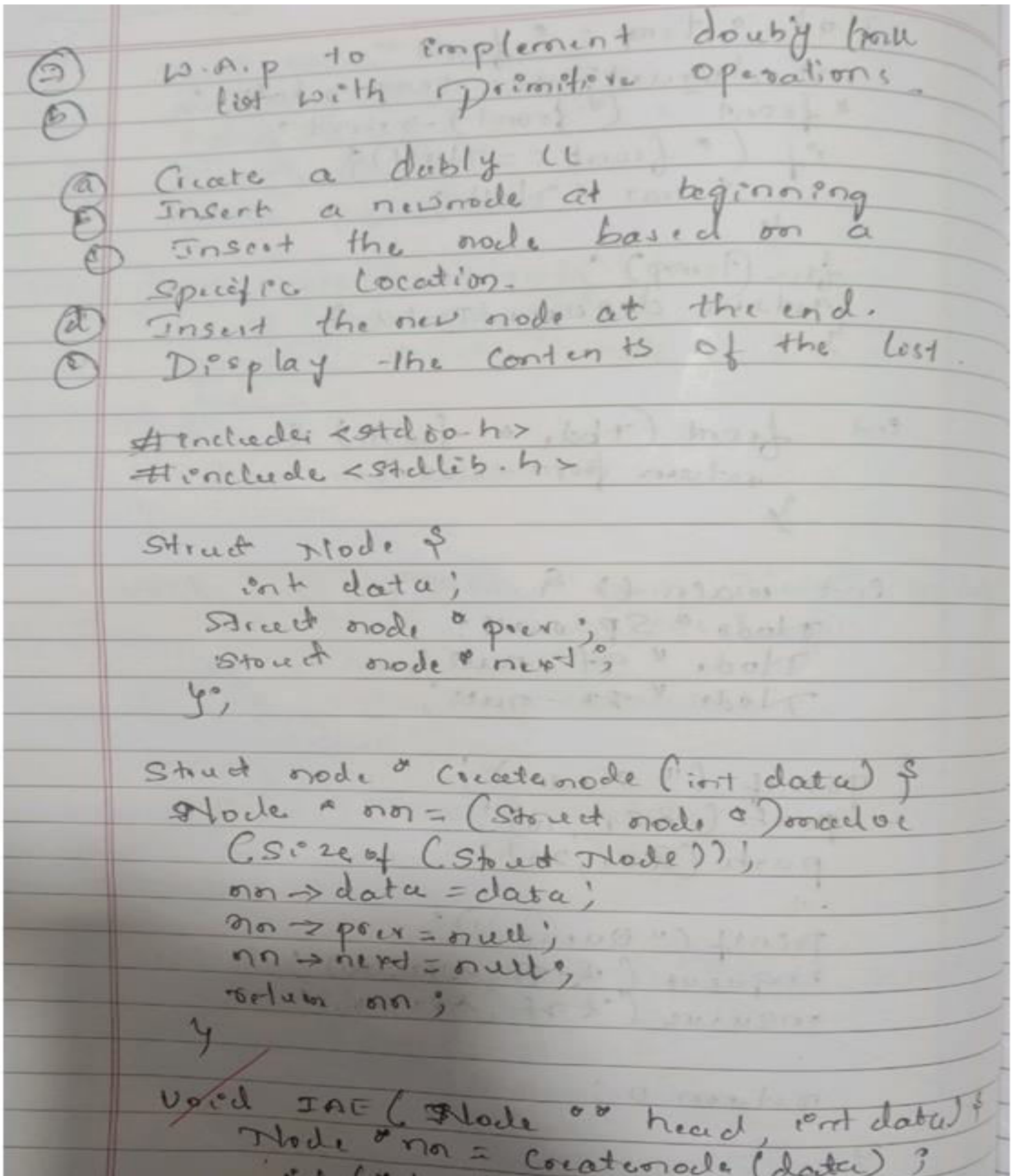
```
            printf("Enter number of nodes for second list: ");
            scanf("%d", &n);
            for (i = 0; i < n; i++) {
                printf("Enter data for node %d of list2: ", i + 1);
                scanf("%d", &data);
                struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
                newNode->data = data;
                newNode->next = list2;
                list2 = newNode;
            }
            concatenateLists(list1, list2);
            break;
        case 5:
            displayList();
            break;
        case 6:
            return 0;
        default:
            printf("Invalid choice!\n");
        }
    }
}
```

Output:

```
srujan  R@SRUJAN MINGW64 ~
$ Singly Linked List Operations:
1. Insert at End
2. Sort List
3. Reverse List
4. Concatenate Two Lists
5. Display List
6. Exit
Enter your choice: 1
Enter data to insert at end: 10

Singly Linked List Operations:
1. Insert at End
2. Sort List
3. Reverse List
4. Concatenate Two Lists
5. Display List
6. Exit
Enter your choice: 1
Enter data to insert at end: 20

Singly Linked List Operations:
1. Insert at End
2. Sort List
3. Reverse List
4. Concatenate Two Lists
5. Display List
6. Exit
Enter your choice: 5
Linked list contents: 10 -> 20 -> NULL

Singly Linked List Operations:
1. Insert at End
2. Sort List
3. Reverse List
4. Concatenate Two Lists
5. Display List
6. Exit
Enter your choice: 2
Linked list sorted.

Singly Linked List Operations:
1. Insert at End
2. Sort List
3. Reverse List
4. Concatenate Two Lists
5. Display List
6. Exit
Enter your choice: 5
Linked list contents: 10 -> 20 -> NULL

Singly Linked List Operations:
1. Insert at End
2. Sort List
3. Reverse List
4. Concatenate Two Lists
5. Display List
6. Exit
Enter your choice: 3
Linked list reversed.

Singly Linked List Operations:
1. Insert at End
2. Sort List
3. Reverse List
4. Concatenate Two Lists
Linked list contents: 5 -> 15 -> 25 -> 30 -> NULL
```

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

Observation:



W.A.P to implement doubly link list with primitive operations.

a) Create a doubly LL
b) Insert a newnode at beginning
c) Insert the node based on a specific location.
d) Insert the new node at the end.
e) Display the contents of the list.

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

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

struct node *createnode (int data) {
 Node *nn = (struct node *)malloc
    (sizeof (struct Node));
    nn -> data = data;
    nn -> prev = null;
    nn -> next = null;
    return nn;
}

void IAF ( Node **head, int data) {
  Node *nn = createnode (data);
    if (*head == null) {
      *head = newnode;
      return;
    }
```

```
struct Node * temp = * head;
  While (temp -> next != null) temp = temp->next;
  temp -> next = newnode;
} nn -> prev = temp;

Void IAL ( Node * head, int data, int la){
  struct Node * nn = CreateNode(data);
    if (loc == 0) {
      IAB (head, data);
    return;
    }
}

Void displayList (struct Node * head) {
  Struct Node * temp = head;
  While (temp != null) {
  printf ("-1.d", temp -> data);
    temp = temp -> next;
  }
}

Int main () {
  Node * head = null;

  IAB (&head, 10);
  IAB (&head, 20);
  IAE (&head, 30);
  AAL (&head, 25, 2);

  display list (head);
  return 0;
  }
```

O/P

1) IAB
2) IAE
3) IAL
4) Display
5 Exit

1) 5
1) 6
1) 7

2) 8
2) 9
2) 12

4) 5 6 7 12 9 8

5) Exitting

3) IAL
Index 3

5 6 7 12 3 9 8

5) Exit

Code:

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

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

struct DoublyLinkedList {
    struct Node* head;
};

void createList(struct DoublyLinkedList* dll, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->prev = newNode->next = NULL;

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

```
}

void insertLeft(struct DoublyLinkedList* dll, int newData, int existingData) {
    struct Node* temp = dll->head;
    while (temp != NULL && temp->data != existingData) {
        temp = temp->next;
    }

    if (temp != NULL) {
        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->data = newData;
        newNode->next = temp;
        newNode->prev = temp->prev;

        if (temp->prev != NULL) {
            temp->prev->next = newNode;
        } else {
            dll->head = newNode;
        }
        temp->prev = newNode;
    } else {
        printf("Node with data %d not found.\n", existingData);
    }
}

void deleteNode(struct DoublyLinkedList* dll, int value) {
    struct Node* temp = dll->head;

    while (temp != NULL && temp->data != value) {
```

```
        temp = temp->next;
    }

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

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

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

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

void display(struct DoublyLinkedList* dll) {
    if (dll->head == NULL) {
        printf("List is empty.\n");
        return;
    }

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

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

int main() {
    struct DoublyLinkedList dll;
    dll.head = NULL;

    createList(&dll, 10);
    createList(&dll, 20);
    createList(&dll, 30);
    createList(&dll, 40);

    printf("Original List:\n");
    display(&dll);

    insertLeft(&dll, 25, 30);
    printf("List after inserting 25 to the left of 30:\n");
    display(&dll);

    deleteNode(&dll, 20);
    printf("List after deleting node with value 20:\n");
    display(&dll);

    deleteNode(&dll, 50);
    return 0;
}
```

Output:

```
srujan  R@SRUJAN MINGW64 ~
$ Original List:
10 20 30 40
List after inserting 25 to the left of 30:
10 20 25 30 40
List after deleting node with value 20:
10 25 30 40
Node with value 50 not found.
```

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.

Observation:

1) To Construct a BST
2) To traverse using 3 traversal methods
3) To display

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

struct node {
  int data;
  struct node * L;
  struct node * R;
};

Struct node * createNode() {
  Struct node * p = (Struct node *) malloc(sizeof
    (Struct node));
}

Struct node * insertion(Struct node * root, int data
  Struct node * n = createNode();
  if (root == Null) return n;
else if(data < root -> data) {
      root -> left = (insertion(root -> left, data));
  }
else if (data > root -> data) {
      root -> right = (insertion(root -> right, data));
  }
}
```

```
void preorder (Struct node * root) {
    if (root != Null) {
        printf (* root -> data);
        preorder (root -> left);
        preorder (root -> right);
    }
}

Void inorder ( Struct node * root) {
    if ( root != null) {
    inorder ( root -> left) ;
    printf (root -> data);
    inorder (root -> right);
    }

Void postorder (Struct node * root) {
    if (root != null) {
    postorder (root -> left);
    postorder (root -> right);
    printf ( root -> data);
}

int main () {
    Struct node * root = Null;
    root = insertion (root, 5);
    root = insertion (root, 6);

    inorder ( root );
    preorder ( root );
    postorder ( root );

    return 0;
}
```

O/P

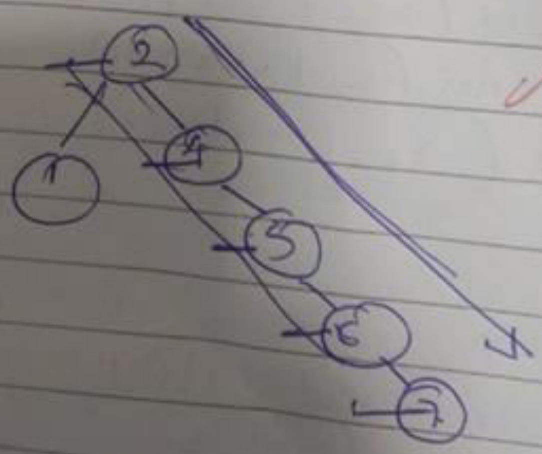
1) Insert
2. preorder
3. Inorder
4. postorder

1) 2 4 5 6 7

2. 2 1 4 5 6 7 : preorder

3. 1 2 4 5 6 7 : Inorder

4. 1 7 6 5 4 2 : postorder



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 = 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 if (data > root->data) {
      root->right = insert(root->right, data);
   }
   return root;
}
```

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

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

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

void display(struct Node* root) {
   if (root != NULL) {
      printf("Inorder: ");
      inorder(root);
      printf("\n");
```

```
        printf("Preorder: ");
        preorder(root);
        printf("\n");

        printf("Postorder: ");
        postorder(root);
        printf("\n");
    }
}

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

    do {
        printf("Binary Search Tree Operations:\n");
        printf("1. Insert Element\n");
        printf("2. Display Elements\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

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

```
        case 2:
          if (root == NULL) {
            printf("Tree is empty.\n");
          } else {
            display(root);
          }
          break;

        case 3:
          printf("Exiting...\n");
          break;

        default:
          printf("Invalid choice, please try again.\n");
      }
  } while (choice != 3);

  return 0;
}
```

Output:

```
srujan  R@SRUJAN MINGW64 ~
$ Binary Search Tree Operations:
1. Insert Element
2. Display Elements
3. Exit
Enter your choice: 1
Enter value to insert: 50

Binary Search Tree Operations:
1. Insert Element
2. Display Elements
3. Exit
Enter your choice: 1
Enter value to insert: 30

Binary Search Tree Operations:
1. Insert Element
2. Display Elements
3. Exit
Enter your choice: 1
Enter value to insert: 70

Binary Search Tree Operations:
1. Insert Element
2. Display Elements
3. Exit
Enter your choice: 2
Inorder: 30 50 70
Preorder: 50 30 70
Postorder: 30 70 50

Binary Search Tree Operations:
1. Insert Element
2. Display Elements
3. Exit
Enter your choice: 3
Exiting...
```

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

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

Observation:

BFS & check my graph is Connected or not

```
void bfs ( int graph[max][max],
    int start, int visited[max] , int
    vertices) {

  struct Queue q ;
  q.front = q.rear = -1 ;
  enqueue (&q, start) ;
  visited (start]=1 ;

  while ( !isEmpty(&q)) {
     int node = dequeue(&q) ;
     printf ("%d", node) ;

   for (int i = 0 ; i < vertices ;i++ ) {
    if ( graph [node] [i] == 1 && !visited[i]){
      visited [i]=1 ;
      enqueue (&q, i) ;
      }
     }
    }
  }
```

```
void dfs (int graph[max][max], int node,
    int visited[max], int vertices) {

    visited[node] = 1;
    printf("%d ", node);

    for (int i=0; i< vertices; i++) {
        if (graph[node][i] == 1 && !visited[i]
            dfs (graph, i, visited, vertices);
    }
}

int isConnected (int graph[max][max],
                 int vertices) {

    int visited[max] = {0};
    dfs (graph, 0, visited, vertices);

    for (int i=0; i< vertices; i++) {
        if (visited[i]==0)
            return 0;
    }
    return 1;
}

int main() {
// defining graph;
Switch cases

}
```

1) BFS

1 5 3 2 6 4

2) DFS

1 3 5 2 6 4

3) is connected
graph is connected.

23/12

Code:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 10
struct Queue {
    int items[MAX];
    int front;
    int rear;
};

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

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

bool isFull(struct Queue* q) {
    return q->rear == MAX - 1;
}

void enqueue(struct Queue* q, int value) {
    if (isFull(q)) {
        printf("Queue is full\n");
        return;
    }
    if (q->front == -1)
```

```
        q->front = 0;
    q->rear++;
    q->items[q->rear] = value;
}

int dequeue(struct Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return -1;
    }
    int item = q->items[q->front];
    q->front++;
    if (q->front > q->rear)
        q->front = q->rear = -1;
    return item;
}

void BFS(int graph[MAX][MAX], int visited[MAX], int start, int n) {
    struct Queue q;
    initQueue(&q);
    enqueue(&q, start);
    visited[start] = 1;

    while (!isEmpty(&q)) {
        int current = dequeue(&q);
        printf("%d ", current);

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

```
            enqueue(&q, i);
            visited[i] = 1;
          }
        }
    }
}


void DFS(int graph[MAX][MAX], int visited[MAX], int vertex, int n) {
    visited[vertex] = 1;
    printf("%d ", vertex);

    for (int i = 0; i < n; i++) {
        if (graph[vertex][i] == 1 && !visited[i]) {
            DFS(graph, visited, i, n);
        }
    }
}

int isConnected(int graph[MAX][MAX], int n) {
    int visited[MAX] = {0};
    DFS(graph, visited, 0, n);

    for (int i = 0; i < n; i++) {
        if (!visited[i]) {
            return 0;  // Not connected
        }
    }
    return 1;  // Connected
```

```
}

int main() {
   int graph[MAX][MAX] = {0};
   int n, edges, u, v;

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

   for (int i = 0; i < edges; i++) {
      printf("Enter edge (u v): ");
      scanf("%d %d", &u, &v);
      graph[u][v] = 1;
      graph[v][u] = 1;  // For undirected graph
   }

   if (isConnected(graph, n)) {
      printf("The graph is connected.\n");
   } else {
      printf("The graph is not connected.\n");
   }

   return 0;
}
```

Output:

```
srujan  R@SRUJAN MINGW64 ~
$ Enter the number of vertices: 5
Enter the number of edges: 4
Enter edge (u v): 0 1
Enter edge (u v): 0 2
Enter edge (u v): 1 3
Enter edge (u v): 2 4
BFS traversal starting from vertex 0: 0 1 2 3 4
```

```
srujan  R@SRUJAN MINGW64 ~
$ Enter the number of vertices: 5
Enter the number of edges: 4
Enter edge (u v): 0 1
Enter edge (u v): 0 2
Enter edge (u v): 1 3
Enter edge (u v): 2 4
0 1 3 2 4
```