VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

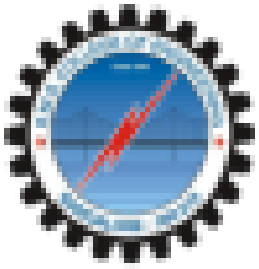
Data Structures using C Lab

(23CS3PCDST)

Submitted by

Uday Shankar Y (1BM24CS427)

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 **Uday Shankar Y(1BM24CS427),** 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.

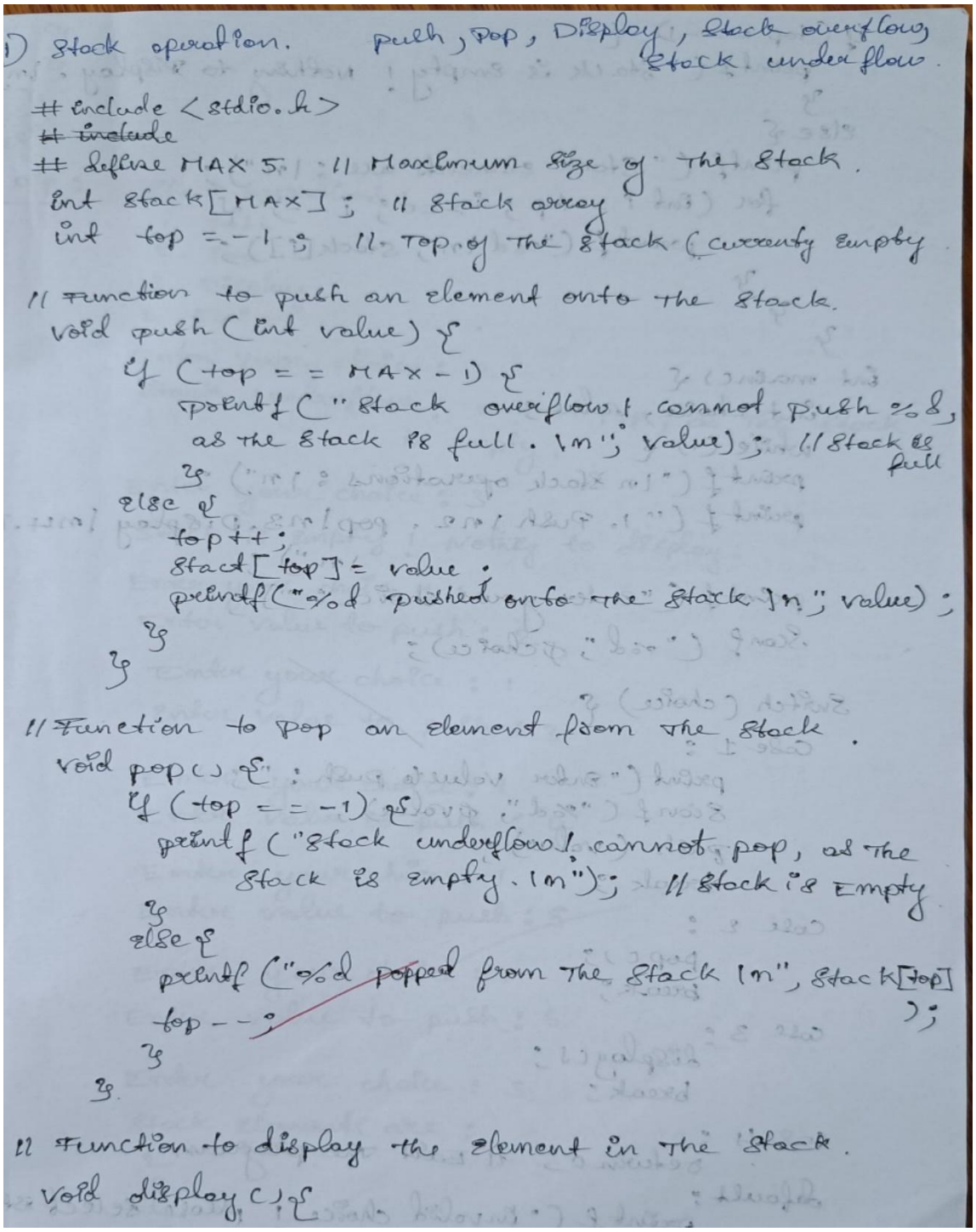
Lab faculty Incharge Name Ramya K M Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

Index

Github Link : https://github.com/UdayShankarY/1BM24CS427-UdayShankarY-DS

Program 1

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



```
1) Stack operation.  Push, Pop, Display, Stack overflow, Stack underflow.

#include <stdio.h>
#include
#define MAX 5  // Maximum size of the stack.
int stack[MAX];  // Stack array
int top = -1;  // Top of the stack (currently empty

// Function to push an element onto the stack.
void push(int value) {
    if (top == MAX - 1) {
        printf("Stack overflow! cannot push %d,
        as the stack is full.\n", value);  // Stack is full
    }
    else {
        top++;
        stack[top] = value;
        printf("%d pushed onto the stack\n", value);
    }
}

// Function to pop an element from the stack.
void pop() {
    if (top == -1) {
        printf("Stack underflow! cannot pop, as the
            stack is empty.\n");  // Stack is Empty
    }
    else {
        printf("%d popped from the stack\n", stack[top]
                                                    );
        top--;
    }
}

// Function to display the element in the stack.
void display() {
```

```
if (top == -1) {
    printf("Stack is empty! Nothing to display.\n");
}
else {
    printf("Stack elements are:\n");
    for (int i = top; i >= 0; i--) {
        printf("%d\n", stack[i]);
    }
}
int main() {
    int choice, value;
    while (1) {
    printf("\n Stack operations:\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(value);
            break;
        case 2:
            pop();
            break;
        case 3:
            display();
            break;
        case 4:
            return 0; // Exit the program
        default:
            printf("Invalid choice! please select between
                    1-4.\n");
```

Seen

O/P :

```
Stack operations :
1. Push
2. Pop
3. Display
4. Exit
Enter your choice : 2
Stack underflow ! cannot pop, as the stack is empty.
Enter your choice : 3
Stack is empty ! Nothing to display.
Enter your choice : 1
Enter value to push : 2
Enter your choice : 1
Enter value to push : 3
Enter your choice : 1
Enter value to push : 4
Enter your choice : 1
Enter value to push : 5
Enter your choice : 1
Enter value to push : 6.
Enter your choice : 3
Stack elements are :
2
3
4
5
6
Enter your choice : 1.
Enter value to push : 8.
Stack overflow ! cannot push 8, as the stack is full
```

Seen
01/10/24

Code :

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

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

void push(int value) {
   if (top == MAX - 1) {
      printf("Stack Overflow! Cannot push %d\n", value);
   } else {
      top++;
      stack[top] = value;
      printf("Pushed %d onto the stack.\n", value);
   }
}

void pop() {
   if (top == -1) {
      printf("Stack Underflow! Cannot pop from an empty stack.\n");
   } else {
      printf("Popped %d from the stack.\n", stack[top]);
      top--;
   }
}

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

int main() {
   int choice, value;

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

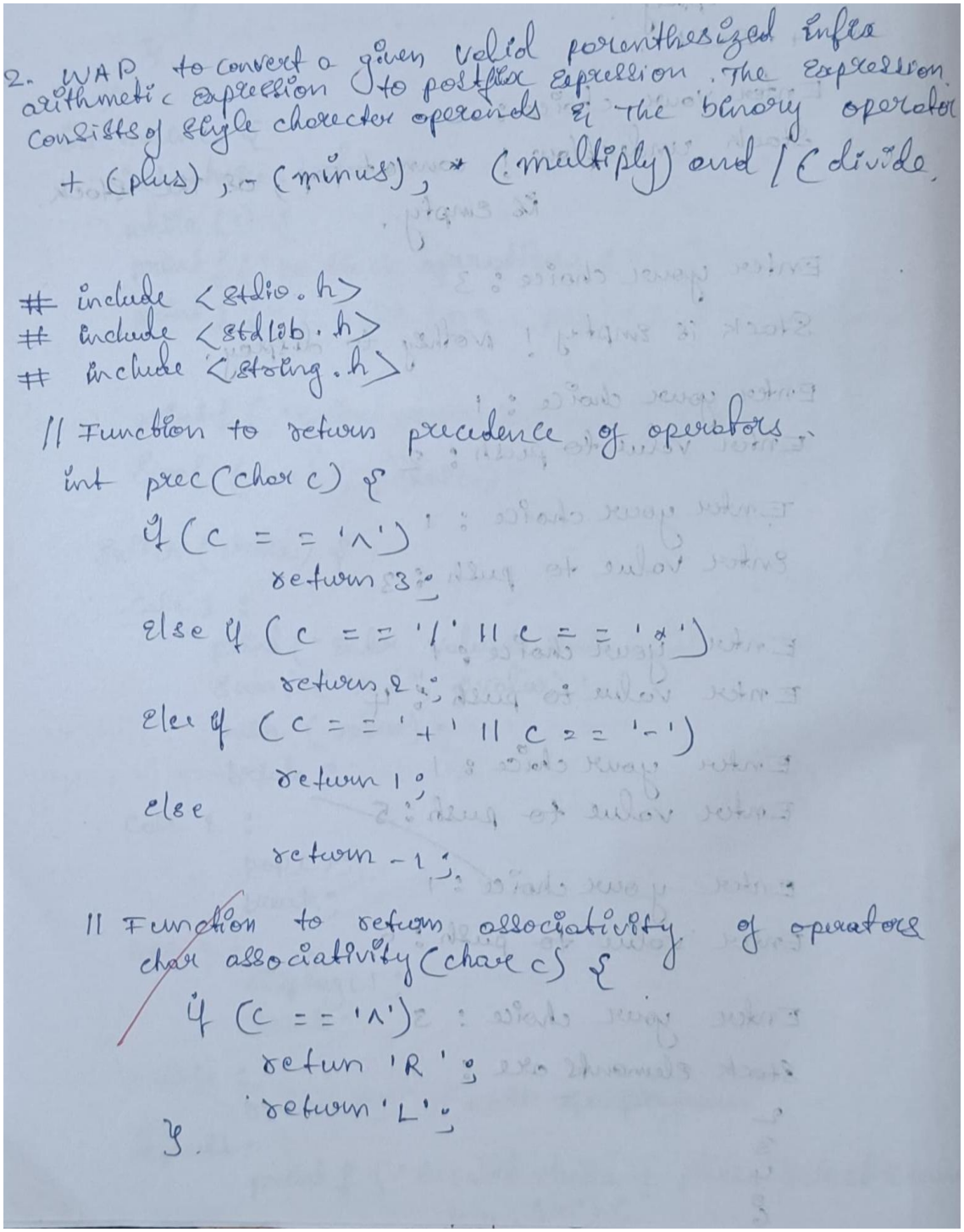
```
switch (choice) {
        case 1:
            printf("Enter the value to push: ");
            scanf("%d", &value);
            push(value);
            break;
        case 2:
            pop();
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting program.\n");
            return 0;
        default:
            printf("Invalid choice! Please enter a number between 1 and 4.\n");
    }
  }
}
```

OUTPUT :

```
Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to push: 25
Pushed 25 onto the stack.

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to push: 10
Pushed 10 onto the stack.

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Popped 10 from the stack.

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

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack elements: 25 15

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 4
Exiting program.
PS C:\Users\Admin\Desktop\1BM23CS314\C>
```

Program 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)



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

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// Function to return precedence of operators.
int prec(char c) {
    if (c == '^')
        return 3;
    else if (c == '/' || c == '*')
        return 2;
    else if (c == '+' || c == '-')
        return 1;
    else
        return -1;
// Function to return associativity of operators
char associativity(char c) {
    if (c == '^')
        return 'R';
    return 'L';
}
```

```
// The main function to convert infix expression to
   postfix expression
void infixToPostfix (const char *s) {
    int len = strlen(s);
// Dynamically allocate memory for result and stack
    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 the scanned character is an operand, add to
    the output string from the stack until an '('
    is encountered.
else if (c == ')') {
    while (stackIndex >= 0 && stack[stackIndex]
            != '(') {
        result[resultIndex++] = stack[stackIndex--];
    }
    stackIndex--; // Pop '('
}
else {      // If an operator is scanned.
    while (stackIndex >= 0 && (prec(c) <
                prec(stack[stackIndex]) ||
    (prec(c) == prec(stack[stackIndex])
    && associativity(c) == 'L'))) {
        result[resultIndex++] = stack[stack
            Index--];
    }
    stack[++stackIndex] = c;
```

```
}
while (stackIndex >= 0) {    // pop all the remaining elements from stack.
    result[resultIndex++] = stack[stackIndex--];
}
result[resultIndex] = '\0';    // Null-terminate the result
printf("%s\n", result);
// Free the allocated memory
free(result);
free(stack); }.

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

Manual

o/p: If exp[] = "a+b*(c^d-e)^(f+g*h)-

~~-> abcd^e-fgh*+^*+i-~~

a+b*(c^d-e)^(f+g*h)-i

a+b*cd^e-^fgh*+-i

cd^e-fgh*+^

a+b*cd^efgh*+^-i

abcd^e-fgh*+^+-i

abcd^e-fgh*+^*+i-

Manual output.

14/10

Code:

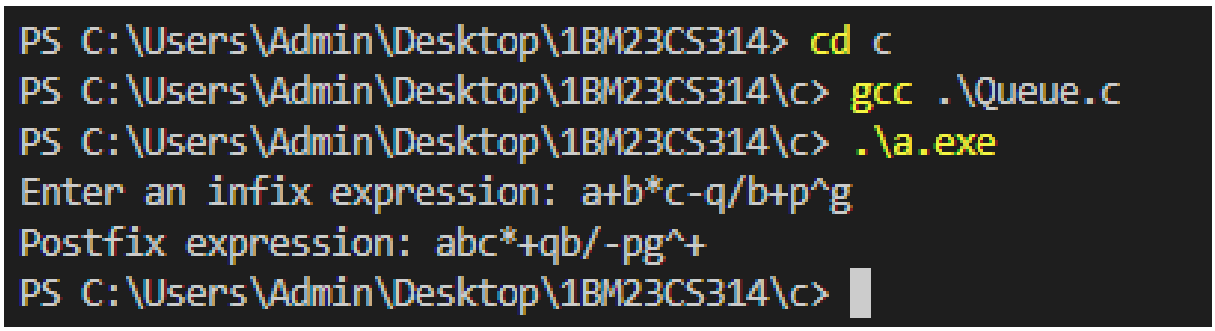
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 100

// Function to return precedence of operators
int prec(char c) {
    if (c == '^')
          return 3;
    else if (c == '/' || c == '*')
         return 2;
    else if (c == '+' || c == '-')
        return 1;
    else
        return -1;
 }




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

void infixToPostfix(const char *s) {
char result[MAX];
char stack[MAX];
int resultIndex = 0;
int stackIndex = -1;
int len = strlen(s);
for (int i = 0; i < len; i++) {
```

```
    char c = s[i];
// If the scanned character is an operand, add it to the output string.
    if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= '0' && c <= '9')) {
            result[resultIndex++] = c;
    } else if (c == '(') {
        // If the scanned character is an '(', push it to the stack.
             stack[++stackIndex] = c;
                                                                                    }
     else if (c == ')') {
// If the scanned character is an ')', pop and add to the output string from the stack until
an '(' is encountered.
        while (stackIndex >= 0 && stack[stackIndex] != '(') {
                result[resultIndex++] = stack[stackIndex--];
                                                                                       }
            stackIndex--; // Pop '('
                                                                                    }
else {
        // If an operator is scanned
        while (stackIndex >= 0 && (prec(c) < prec(stack[stackIndex]) ||
                  (prec(c) == prec(stack[stackIndex]) && associativity(c) == 'L'))) {
                result[resultIndex++] = stack[stackIndex--];
                                                                                    }
            stack[++stackIndex] = c;
                                                                                    }
}
```

```
// Pop all the remaining elements from the stack
while (stackIndex >= 0) {
        result[resultIndex++] = stack[stackIndex--];
}

    result[resultIndex] = '\0';   // Null-terminate the result
```

```
    printf("Postfix expression: %s\n", result);
}

int main() {
char exp[MAX];

printf("Enter an infix expression: ");
fgets(exp, MAX, stdin);
exp[strcspn(exp, "\n")] = 0; // Remove trailing newline

    infixToPostfix(exp);
return 0;
}
```

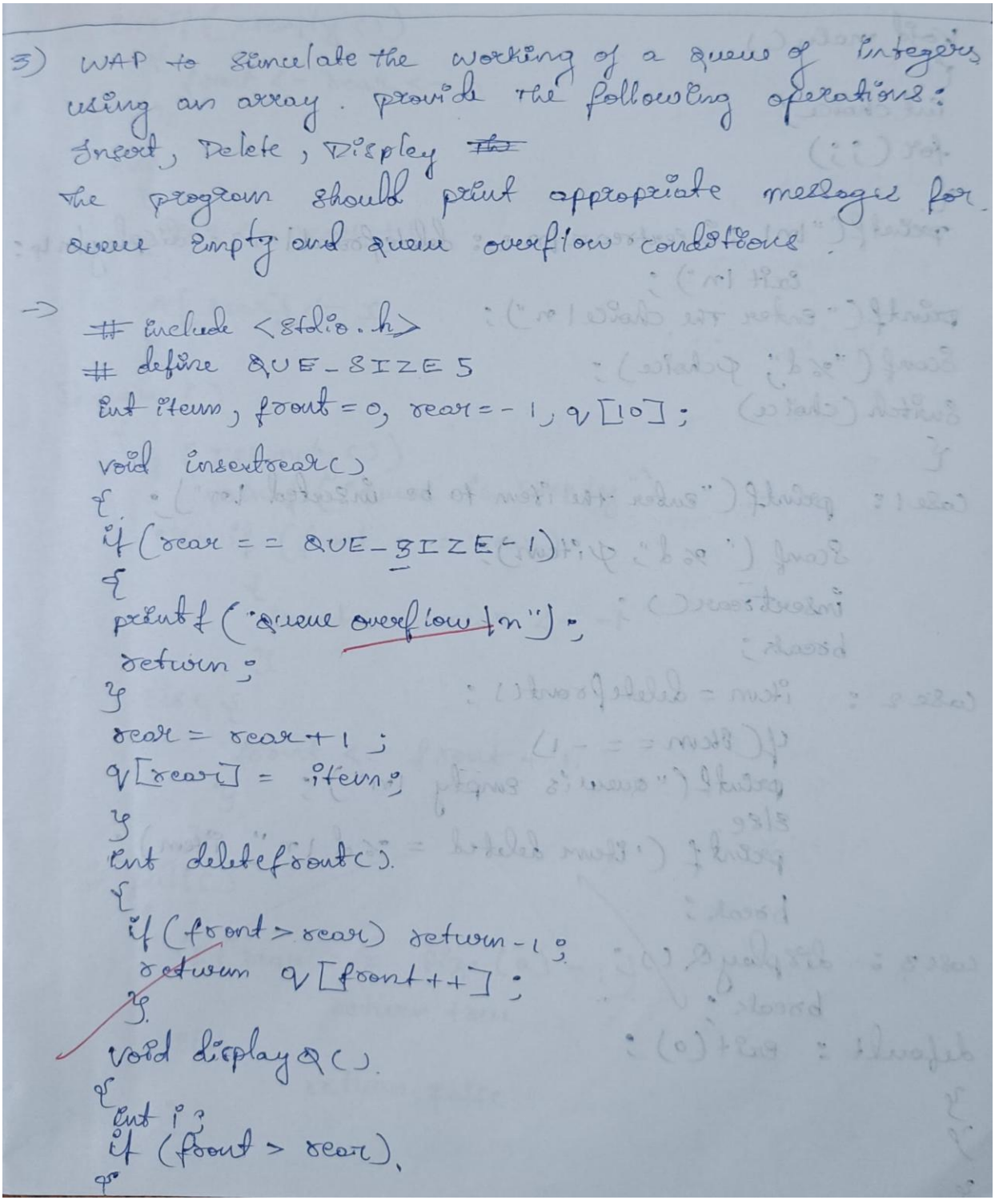
Output:



Program 3:

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



3) 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

→

```
#include <stdio.h>
#define QUE_SIZE 5
int item, front=0, rear=-1, q[10];
void insertrear()
{
if(rear == QUE_SIZE-1);
{
printf("queue overflow\n");
return;
}
rear = rear+1;
q[rear] = item;
}
int deletefront().
{
if(front>rear) return-1;
return q[front++];
}
void displayQ().
{
int i;
if (front > rear),
```

```
printf("queue is empty \n");
return;
}
printf("contents of queue\n");
for(i=front; i<=rear; i++)
{
printf("%d \n", q[i]);
}
}

void main()
{
int choice;
for(;;)
{
printf("\n1 : insertrear\n2 : deletefront\n3 : display\n4:
        exit\n");
printf("enter the choice\n");
scanf("%d", &choice);
switch(choice)
{
case 1: printf("enter the item to be inserted \n");
        scanf("%d", &item);
        insertrear();
        break;
case 2 : item = deletefront();
         if(item == -1)
         printf("queue is empty \n");
         else
         printf("item deleted = %d \n", item);
         break;
case 3 : display Q();
         break;
default : exit(0);
}
}
```

o/p :

1. Enqueue
2. Dequeue
3. display
4. exit

Enter your choice : 2 // Trying to Dequeue when its empty
Queue is empty ... !

Enter your choice : 3 // Trying to Display when its empty
Queue is empty ... !

Enter your choice : 1. // Inserting an element
3.

Enter your choice : 3. // Displaying after inserting an element
3

Enter your choice : 1 // 2nd time
2

Enter your choice : 1 // 3rd time
5

Enter your choice : 1 // 4th time
7

Enter your choice : 1 // 5th time
9.

Enter your choice : 1 // Trying to insert 5th element.
Queue is full ... !

Enter your choice : 3. // Displaying Full queue
3 2 5 7 9.

Enter your choice : 2. // Dequeuing
Enter your choice : 3. // Displaying after dequeuing
2 5 7 9.

14/10

Code:

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

#define MAX 5

// Function to check if the queue is full
int isFull(int rear) {
    if (rear == MAX - 1) {
        return 1;
    }
    return 0;
}

// Function to check if the queue is empty
int isEmpty(int front, int rear) {
    if (front == -1 || front > rear) {
        return 1;
    }
    return 0;
}

// Function to insert an element into the queue
void insert(int queue[], int *front, int *rear, int value) {
    if (isFull(*rear)) {
        printf("Queue Overflow! Cannot insert %d\n", value);
        return;
    }

    if (*front == -1) {
        *front = 0;  // Set front to 0 when inserting the first element
    }

    (*rear)++;
    queue[*rear] = value;
    printf("%d inserted into the queue\n", value);
}



// Function to delete an element from the queue
```

```
void delete(int queue[], int *front, int *rear) {
    if (isEmpty(*front, *rear)) {
        printf("Queue Underflow! No element to delete\n");
        return;
    }

    int deletedValue = queue[*front];
    printf("%d deleted from the queue\n", deletedValue);
    (*front)++;

    // Reset the queue if it becomes empty
    if (*front > *rear) {
        *front = *rear = -1;
    }
}

// Function to display the elements of the queue
void display(int queue[], int front, int rear) {
    if (isEmpty(front, rear)) {
        printf("Queue is empty!\n");
        return;
    }

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

```
// Main function
int main() {
    int queue[MAX]; // Queue array
    int front = -1, rear = -1; // Initialize front and rear to -1

    int choice, value;

    while (1) {
```

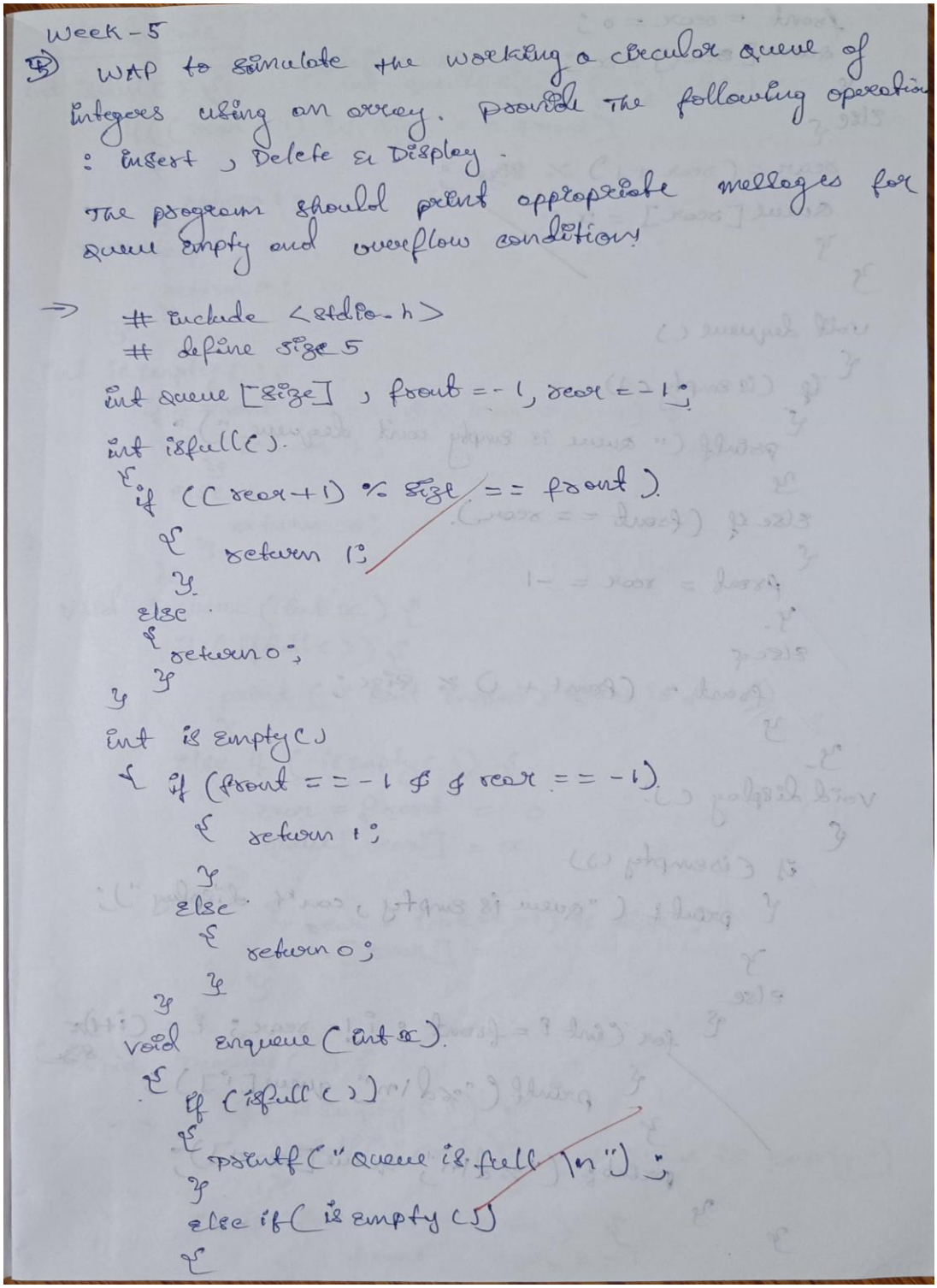
```
        printf("\nQueue Operations:\n");
        printf("1. Insert\n");
        printf("2. Delete\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

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

            case 2:
                delete(queue, &front, &rear);
                break;

            case 3:
                display(queue, front, rear);
                break;

            case 4:
                exit(0);

            default:
                printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}
```

Output:

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

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 25
25 inserted into the queue

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
16 deleted from the queue

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue elements: 25

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 4
PS C:\Users\Shashank U\Desktop\C>
```

Program 4:

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



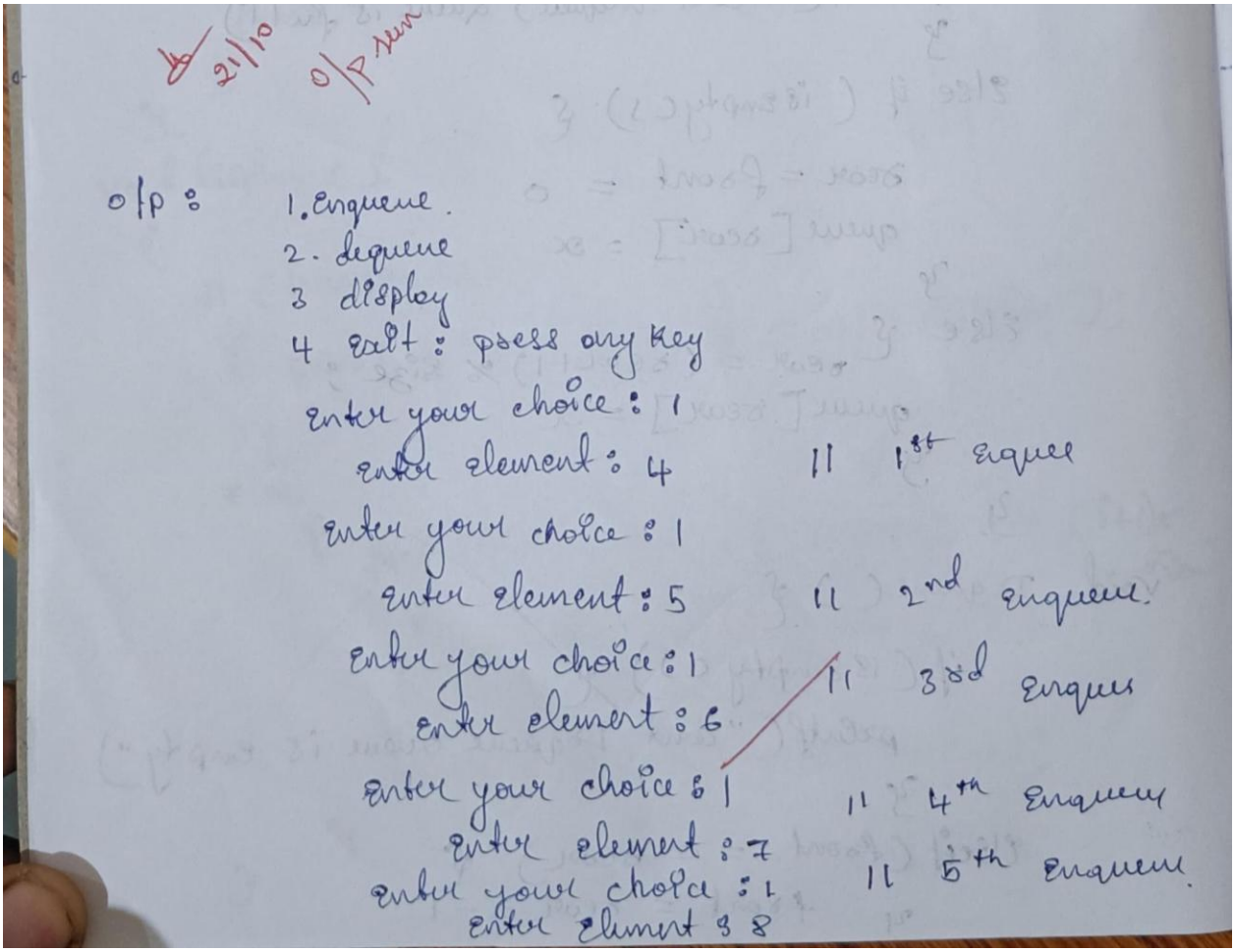
Week-5

5) WAP to simulate the working 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 overflow conditions

→

```
#include <stdio.h>
#define size 5
int queue[size], front = -1, rear = -1;
int isfull().
{ if ((rear+1) % size == front)
  { return 1;
  }
  else
  { return 0;
  }
}
int isempty()
{ if (front == -1 && rear == -1)
  { return 1;
  }
  else
  { return 0;
  }
}
void enqueue (int x).
{ if (isfull())
  { printf("queue is full \n");
  }
  else if(isempty())
  {
```

```
    front = rear = 0;
    queue[rear] = x;
  }
  else {
    rear = (rear + 1) % size;
    queue[rear] = x;
  }
}

void dequeue()
{
  if (isempty())
  {
    printf("queue is empty can't dequeue");
  }
  else if (front == rear)
  {
    front = rear = -1
  }
  else
    front = (front + 1) % size;
  }
}
void display()
{
  if (isempty())
  { printf("queue is empty, can't display");
  }
  else
  { for (int i = front; i != rear; i = (i+1) % size)
    { printf("%d\n", queue[i]);
    }
    printf("%d\n", queue[rear]);
  }
}
```



Code:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5 // Maximum size of the circular queue
// Function to check if the queue is full
int isFull(int front, int rear) {
    if ((rear + 1) % MAX == front) {
        return 1;
    }
    return 0;
}
// Function to check if the queue is empty
int isEmpty(int front, int rear) {
    if (front == -1) {
        return 1;
    }
```

```
    return 0;
}
// Function to insert an element into the queue
void insert(int queue[], int *front, int *rear, int value) {
    if (isFull(*front, *rear)) {
        printf("Queue Overflow! Cannot insert %d\n", value);
        return;
    }
    // If the queue is empty
    if (*front == -1) {
        *front = *rear = 0;
    } else {
        // Circular increment of rear
        *rear = (*rear + 1) % MAX;
    }
    queue[*rear] = value;
    printf("%d inserted into the queue\n", value);
}
// Function to delete an element from the queue
void delete(int queue[], int *front, int *rear) {
    if (isEmpty(*front, *rear)) {
        printf("Queue Underflow! No element to delete\n");
        return;
    }
    int deletedValue = queue[*front];
    printf("%d deleted from the queue\n", deletedValue);
    // If there is only one element left in the queue
    if (*front == *rear) {
        *front = *rear = -1; // Queue is now empty
    } else {
        // Circular increment of front
        *front = (*front + 1) % MAX;
    }
}
// Function to display the elements of the queue
void display(int queue[], int front, int rear) {
    if (isEmpty(front, rear)) {
        printf("Queue is empty!\n");
        return;
    }
    printf("Queue elements: ");
    if (front <= rear) {
```

```
        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");
}
// Main function
int main() {
    int queue[MAX]; // Queue array
    int front = -1, rear = -1; // Initialize front and rear to -1
    int choice, value;
    while (1) {
        printf("\nCircular 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 the value to insert: ");
                scanf("%d", &value);
                insert(queue, &front, &rear, value);
                break;
            case 2:
                delete(queue, &front, &rear);
                break;
            case 3:
                display(queue, front, rear);
                break;
            case 4:
                exit(0);
            default:
```

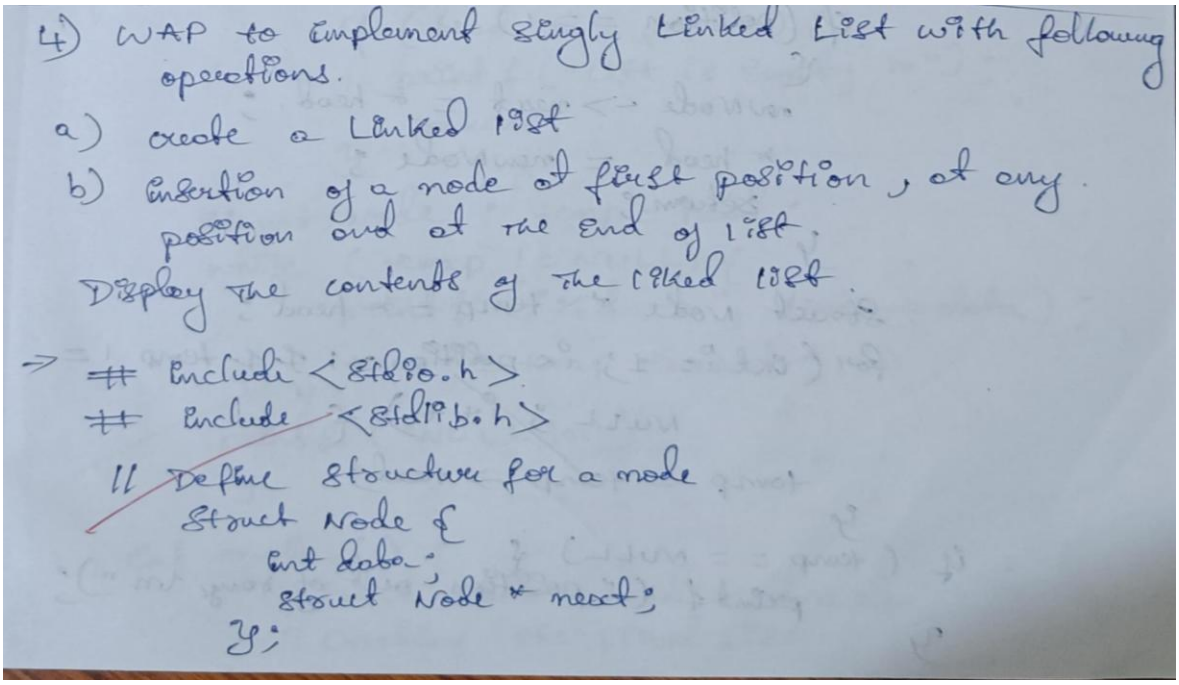
```
            printf("Invalid choice! Please try again.\n");
        }
    }
    return 0;
}
```

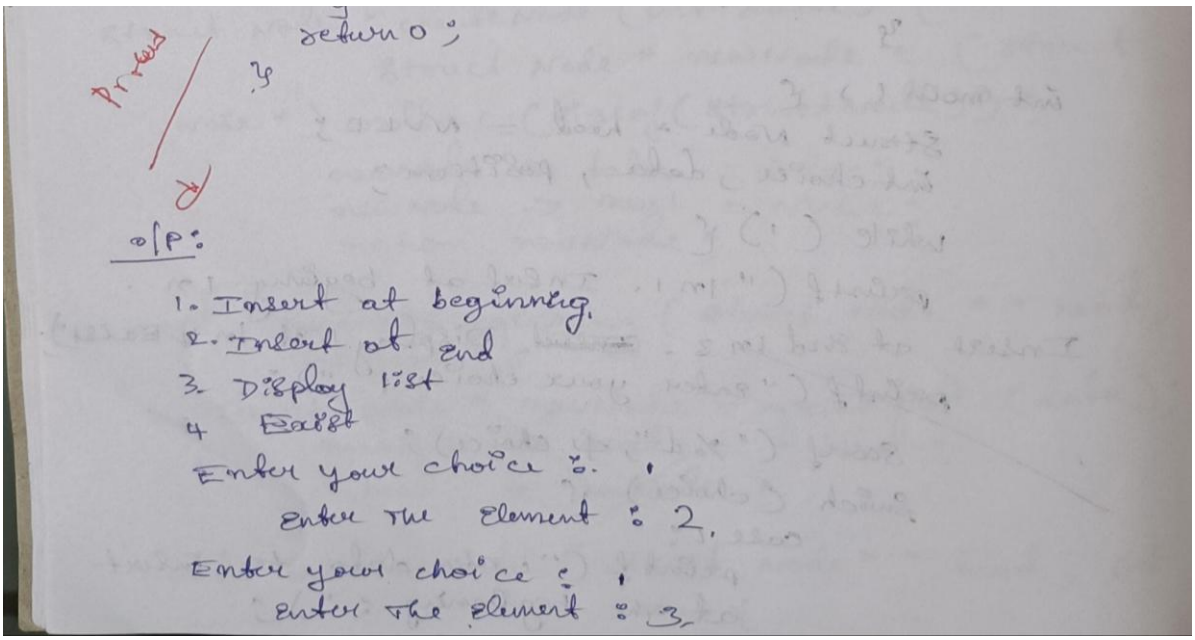
Output:

```
Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 12
12 inserted into the queue

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

Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
12 deleted from the queue

Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue elements: 15

Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 4
PS C:\Users\Shashank U\Desktop\C>
```

Program 5:

05(a) WAP to Implement Singly Linked List with following operations
a) Create a linked list.
b) Insertion of a node at **first position** and **at end of list.**
Display the contents of the linked list.
5b) Leetcode problem no.20 (Valid parantheses)





```
// Function to create a linked list node.
struct Node * createNode (int data) {
    struct Node * newNode = (struct Node *)
        sizeof (struct Node));
    newNode -> data = data;
    newNode -> next = NULL;
    return newNode;
}
// Function to insert at the beginning
void InsertAtBeginning (struct Node ** head, int data)
{ struct Node * newNode = createNode (data);
  newNode -> next = *head;
  *head = newNode;
}
// Function to insert at a specified position
void InsertAtposition (struct Node ** head, int
    data, int position) {
    struct Node * newNode = createNode (data);
    if (position == 1)
    {
      newNode -> next = *head;
      *head = newNode;
      return;
    }
    struct Node * temp = *head;
    for (int i = 1; i < position - 1 && temp !=
            NULL; i++) {
        temp = temp -> next;
    }
    if (temp == NULL) {
        printf ("position out of range \n");
    }
```

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

// Function to insert at the End.
void insertEnd (Struct Node ** head, int data) {
    Struct Node * newNode = create Node (data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    Struct Node * temp = *head;
    while (temp -> next != NULL) {
        temp = temp -> next;
    }
    temp -> next = newNode;
}

// Function to display the linked list
void ~~display the linked list~~
void displayList (Struct Node * head) {
    if (head == NULL) {
        printf ("List is Empty \n");
        return;
    }
    Struct Node * temp = head;
    while (temp != NULL) {
        printf ("%d -> ", temp -> data);
        temp = temp -> next;
    }
    printf ("NULL \n");
}

int main () {
    Struct Node * head = NULL;
    // Creating the Linked List
```

```
InsertAtEnd(&head, 10);
InsertAtEnd(&head, 20);
InsertAtEnd(&head, 30);
displayList(head);

InsertAtBeginning(&head, 5);
displayList(head);
InsertAtPosition(&head, 15, 3);
displayList(head);
InsertAtEnd(&head, 40);
displayList(head);
return 0;
}
```

o/p:

```
1. Insert at beginning
2. Insert at end
3. Display list
4 Exist
Enter your choice : 1
    Enter the Element : 2
Enter your choice : 1
    Enter the Element : 3
```

```
Enter the choice : 2
    Enter element : 9
Enter the choice : 3
    contents of the linked list : 2 -> 9 -> Null
Enter the choice : 4
```

o/p Scan

Code:

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

// Define the structure for a linked list node
struct Node {
    int data;
    struct Node* next;
};

// Function to create a linked list
void createList(struct Node** head) {
    *head = NULL;
}

// Function to insert a node at the beginning
void insertAtFirst(struct Node** head, int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = *head;
    *head = newNode;
    printf("Node with value %d inserted at the beginning.\n", value);
}

// Function to insert a node at any position
void insertAtPosition(struct Node** head, int value, int position) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;

    if (position == 1) {
        newNode->next = *head;
        *head = newNode;
        printf("Node with value %d inserted at position 1.\n", value);
        return;
    }

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

```
    if (temp == NULL) {
        printf("Position %d is out of bounds. Insertion failed.\n", position);
    } else {
        newNode->next = temp->next;
        temp->next = newNode;
        printf("Node with value %d inserted at position %d.\n", value, position);
    }
}

// Function to insert a node at the end of the list
void insertAtEnd(struct Node** head, int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;

    if (*head == NULL) {
        *head = newNode;
        printf("Node with value %d inserted at the end.\n", value);
        return;
    }

    struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
    printf("Node with value %d inserted at the end.\n", value);
}

// Function to display the contents of the linked list
void displayList(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");
}

// Main function
int main() {
    struct Node* head;
    createList(&head); // Create an empty list

    int choice, value, position;

    while (1) {
        printf("\nLinked List Operations:\n");
        printf("1. Insert at First\n");
        printf("2. Insert at Position\n");
        printf("3. Insert at End\n");
        printf("4. Display the list\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to insert at the beginning: ");
                scanf("%d", &value);
                insertAtFirst(&head, value);
                break;

            case 2:
                printf("Enter the value to insert: ");
                scanf("%d", &value);
                printf("Enter the position to insert at: ");
                scanf("%d", &position);
                insertAtPosition(&head, value, position);
                break;

            case 3:
                printf("Enter the value to insert at the end: ");
                scanf("%d", &value);
                insertAtEnd(&head, value);
                break;

            case 4:
```

```
            displayList(head);
            break;

        case 5:
            exit(0);

        default:
            printf("Invalid choice! Please try again.\n");
    }
  }

  return 0;
}
```

Output:

```
Linked List Operations:
1. Insert at First
2. Insert at Position
3. Insert at End
4. Display the list
5. Exit
Enter your choice: 1
Enter the value to insert at the beginning: 7
Node with value 7 inserted at the beginning.

Linked List Operations:
1. Insert at First
2. Insert at Position
3. Insert at End
4. Display the list
5. Exit
Enter your choice: 2
Enter the value to insert: 8
Enter the position to insert at: 2
Node with value 8 inserted at position 2.

Linked List Operations:
1. Insert at First
2. Insert at Position
3. Insert at End
4. Display the list
5. Exit
Enter your choice: 3
Enter the value to insert at the end: 18
Node with value 18 inserted at the end.

Linked List Operations:
1. Insert at First
2. Insert at Position
3. Insert at End
4. Display the list
5. Exit
Enter your choice: 4
Linked List: 7 -> 8 -> 18 -> NULL
```

Leetcode problem:

C Auto

```c
bool isValid(char* s) {
    char stack[strlen(s)];
    int top = -1;
    for (int i = 0; s[i] != '\0'; i++) {
        char current = s[i];
        if (current == '(' || current == '{' || current == '[') {
            stack[++top] = current;
        } else {
            if (top == -1) return false; // Stack is empty, invalid string

            char last = stack[top];
            if ((current == ')' && last == '(') ||
                (current == '}' && last == '{') ||
                (current == ']' && last == '[')) {
                top--; // Pop the stack
            } else {
                return false; // Mismatched bracket
            }
        }
    }
    return top == -1; // Valid if stack is empty
}
```

Program 6:

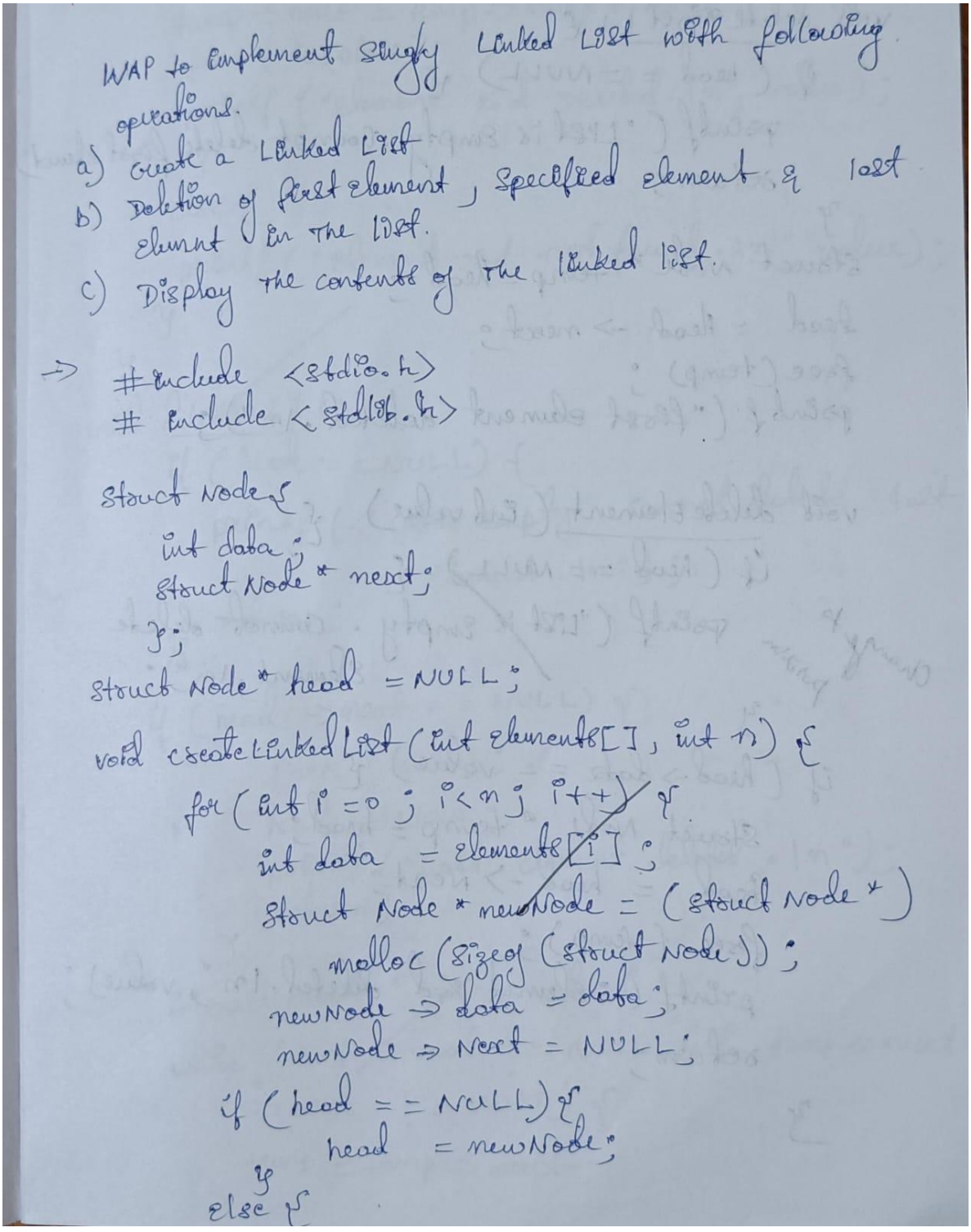
WAP to Implement Singly Linked List with following operations

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.

6b. Leetcode Problem -- 739 (Daily Temperature)



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

void deleteFirst() {
    if (head == NULL) {
        printf("List is empty. Cannot delete first element");
        return;
    }
    struct node * temp = head;
    head = head -> next;
    free(temp);
    printf("first element deleted.\n");

void deleteElement(int value) {
    if (head == NULL) {
        printf("List is empty. Cannot delete
                element.\n");
    }
    if (head -> data == value) {
        struct Node * temp = head;
        head = head -> next;
        free(temp);
        printf("element %d deleted.\n", value);
        return;
    }
```

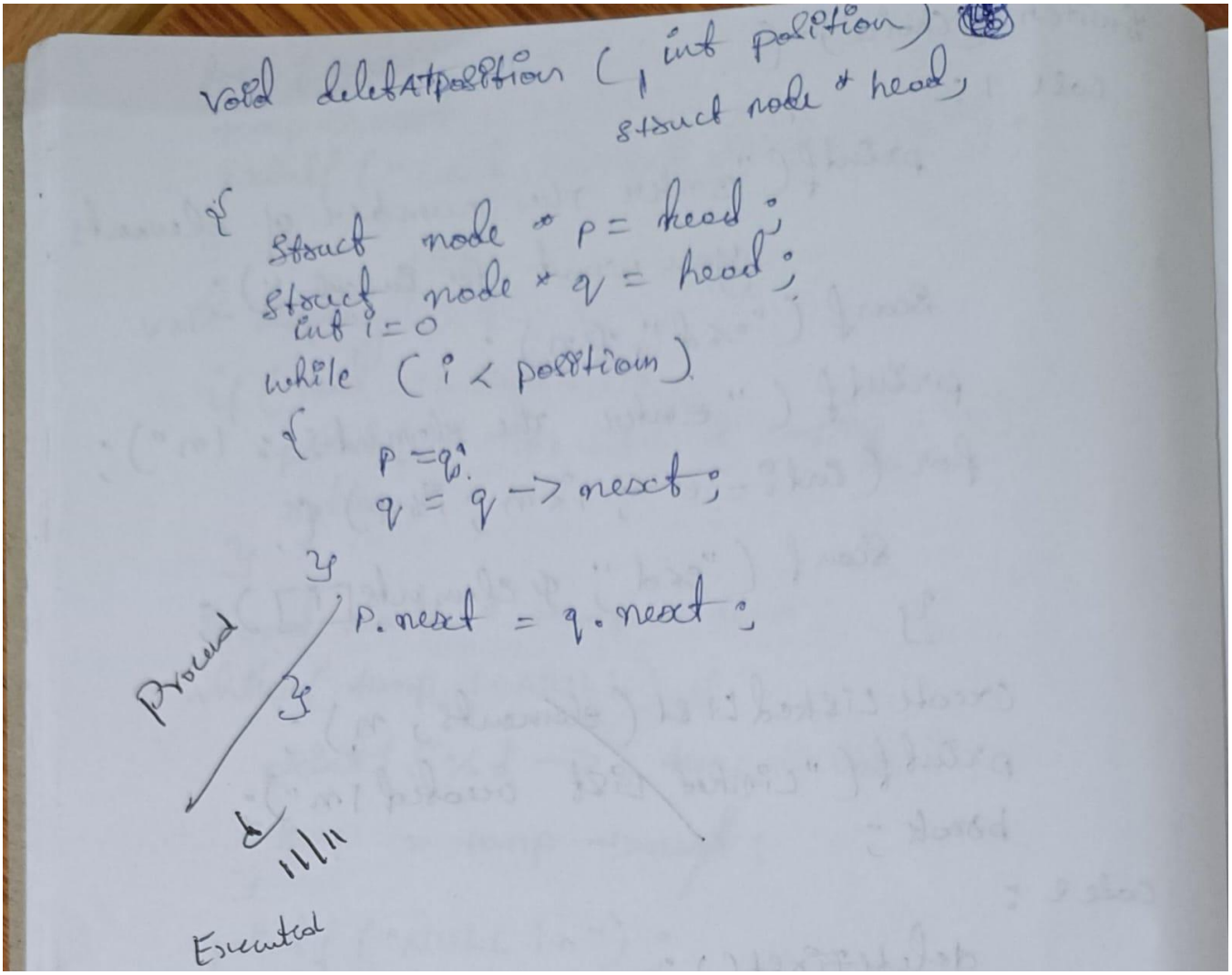
Change to position

10 20

```
struct Node * temp = head;
while (temp -> next != NULL && temp -> next -> data
                != value) {
    temp = temp -> next;
}
if (temp -> next != NULL) {
    struct Node * toDelete = temp -> next;
    temp -> next = temp -> next -> next;
    free (toDelete);
    printf ("Element %d deleted. \n", value);
}
else {
    printf ("Element %d not found. \n", value);
}
}
void deleteLast() {
    if (head == NULL) {
        printf ("List is empty. cannot delete last
                element. \n");
        return;
    }
    if (head -> next == NULL) {
        free (head);
        head = NULL;
        printf ("Last element deleted. \n");
        return;
    }
    struct Node * temp = head
    while (temp -> next != NULL && temp -> next
        -> next != null) {
        temp = temp -> next;
    }
```

```
    free(temp->next);
    temp->next = NULL;
    printf("Last element deleted.\n");
}
void display() {
    if (head == NULL) {
        printf("The List is empty.\n");
        return;
    }
    struct Node *temp = head;
    while (temp != NULL) {
        printf("%d ->", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
    int choice, value, n;
    int elements[10];
                    -> struct node *Head = NULL
    while (1) {
        printf("\nMenu:\n");
        printf("1. Create Linked List");
        printf("2. Delete the first element");
        printf("3. Delete a specified element");
        printf("4. Delete the last element");
        printf("5. Display the linked list\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
switch (choice) {
  case 1:
        printf("Enter the number of elements
                you want to insert:");
        scanf("%d", &n);
        printf("Enter the elements: \n");
        for (int i=0; i<n; i++) {
          scanf("%d", &elements[i]);
        }
        createLinkedList(elements, n);
        printf("Linked List created \n");
        break;
  case 2:
        deleteFirst();
        break;
  case 3:
        printf("Enter the element to delete: ");
        scanf("%d", &value);
        deleteElement(value);
        break;
  case 4:
        deleteLast();
        break;
  case 5:
        displayList();
        break;
  case 6:
        printf("Exiting...\n");
        exit(0);
  default:
        printf("Invalid choice!\n");
    }
    return 0;
}
```



Code:

```
#include <stdio.h>
#include <stdlib.h>
// Define the structure for a linked list node
struct Node {
    int data;
    struct Node* next;
};
// Function to create a linked list
void createList(struct Node** head) {
    *head = NULL;
}
// Function to delete the first element from the list
void deleteFirst(struct Node** head) {
    if (*head == NULL) {
        printf("The list is empty. No element to delete.\n");
```

```
        return;
    }
    struct Node* temp = *head;
    *head = (*head)->next;
    free(temp);
    printf("First element deleted successfully.\n");
}
// Function to delete a specified element from the list
void deleteElement(struct Node** head, int value) {
    if (*head == NULL) {
        printf("The list is empty. No element to delete.\n");
        return;
    }
    struct Node* temp = *head;
    struct Node* prev = NULL;
    // If the element to delete is at the head
    if (temp != NULL && temp->data == value) {
        *head = temp->next;
        free(temp);
        printf("Element %d deleted successfully.\n", value);
        return;
    }
    // Search for the element to delete
    while (temp != NULL && temp->data != value) {
        prev = temp;
        temp = temp->next;
    }
    // If the element was not found
    if (temp == NULL) {
        printf("Element %d not found in the list.\n", value);
        return;
    }
    // Delete the node
    prev->next = temp->next;
    free(temp);
    printf("Element %d deleted successfully.\n", value);
}
// Function to delete the last element from the list
void deleteLast(struct Node** head) {
    if (*head == NULL) {
        printf("The list is empty. No element to delete.\n");
        return;
```

```
    }
    // If there is only one node
    if ((*head)->next == NULL) {
        free(*head);
        *head = NULL;
        printf("Last element deleted successfully.\n");
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL && temp->next->next != NULL) {
        temp = temp->next;
    }
    free(temp->next);
    temp->next = NULL;
    printf("Last element deleted successfully.\n");
}
// Function to display the contents of the linked list
void displayList(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");
}
// Function to insert an element at the end of the list
void insertAtEnd(struct Node** head, int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;
    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Node* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
```

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

// Main function
int main() {
    struct Node* head;
    createList(&head); // Create an empty list

    int choice, value;
    while (1) {
        printf("\nSingly Linked List Operations:\n");
        printf("1. Insert at End\n");
        printf("2. Delete First Element\n");
        printf("3. Delete Specified Element\n");
        printf("4. Delete Last Element\n");
        printf("5. Display the list\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to insert at the end: ");
                scanf("%d", &value);
                insertAtEnd(&head, value);
                break;

            case 2:
                deleteFirst(&head);
                break;

            case 3:
                printf("Enter the value to delete: ");
                scanf("%d", &value);
                deleteElement(&head, value);
                break;

            case 4:
                deleteLast(&head);
                break;
```

```
            case 5:
                displayList(head);
                break;

            case 6:
                exit(0);

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

Output:

```
Singly Linked List Operations:
1. Insert at End
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display the list
6. Exit
Enter your choice: 5
Linked List: 12 -> 5 -> 18 -> 1478 -> 14

Singly Linked List Operations:
1. Insert at End
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display the list
6. Exit
Enter your choice: 2
First element deleted successfully.

Singly Linked List Operations:
1. Insert at End
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display the list
6. Exit
Enter your choice: 3
Enter the value to delete: 14
Element 14 deleted successfully.

Singly Linked List Operations:
1. Insert at End
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display the list
6. Exit
Enter your choice: 4
Last element deleted successfully.
```

```
Singly Linked List Operations:
1. Insert at End
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display the list
6. Exit
Enter your choice: 5
Linked List: 5 -> 18 -> 1478 -> NULL

Singly Linked List Operations:
1. Insert at End
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display the list
6. Exit
Enter your choice: 3
Enter the value to delete: 1475
Element 1475 not found in the list.

Singly Linked List Operations:
1. Insert at End
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display the list
6. Exit
Enter your choice: 6
PS C:\Users\Shashank U\Desktop\C>
```

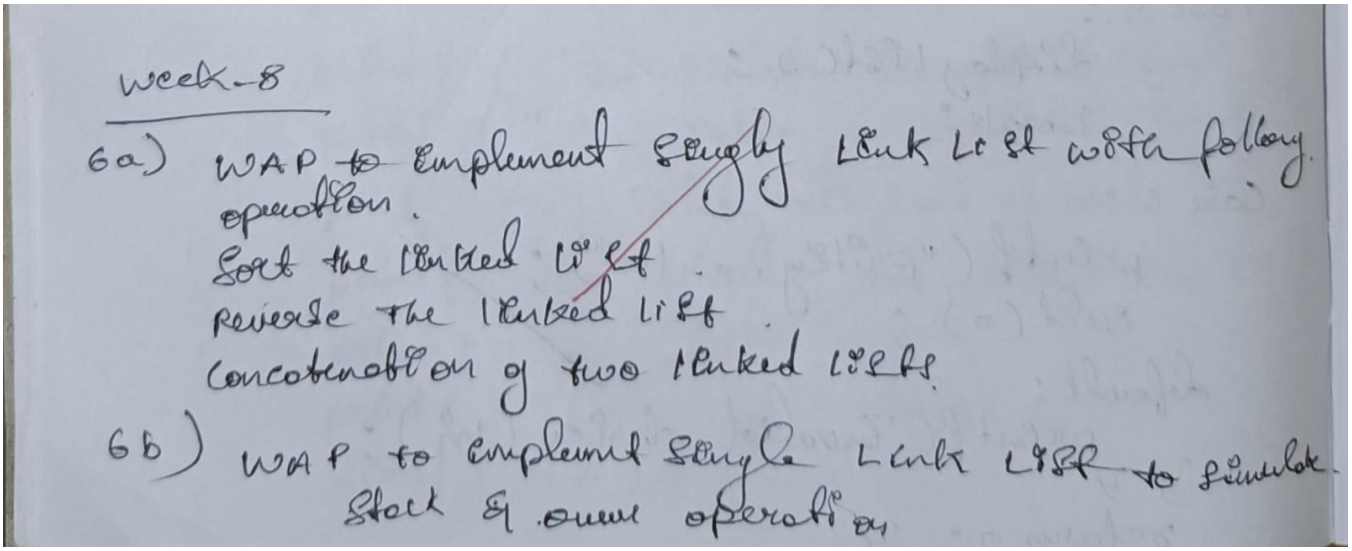
Leetcode problem(Daily Temperatures):

C Auto

```c
int* dailyTemperatures(int* temperatures, int temperaturesSize, int* returnSize) {
    *returnSize=temperaturesSize;
    int* answer=(int*)calloc(temperaturesSize,sizeof(int));
    int* stack = (int*)malloc(temperaturesSize * sizeof(int));
    int top = -1;

    for (int i = 0; i < temperaturesSize; i++) {
        while (top >= 0 && temperatures[i] > temperatures[stack[top]]) {
            int idx = stack[top--];
            answer[idx] = i - idx;
        }
        stack[++top] = i;
    }

    *returnSize = temperaturesSize;
    free(stack);
    return answer;
}
```

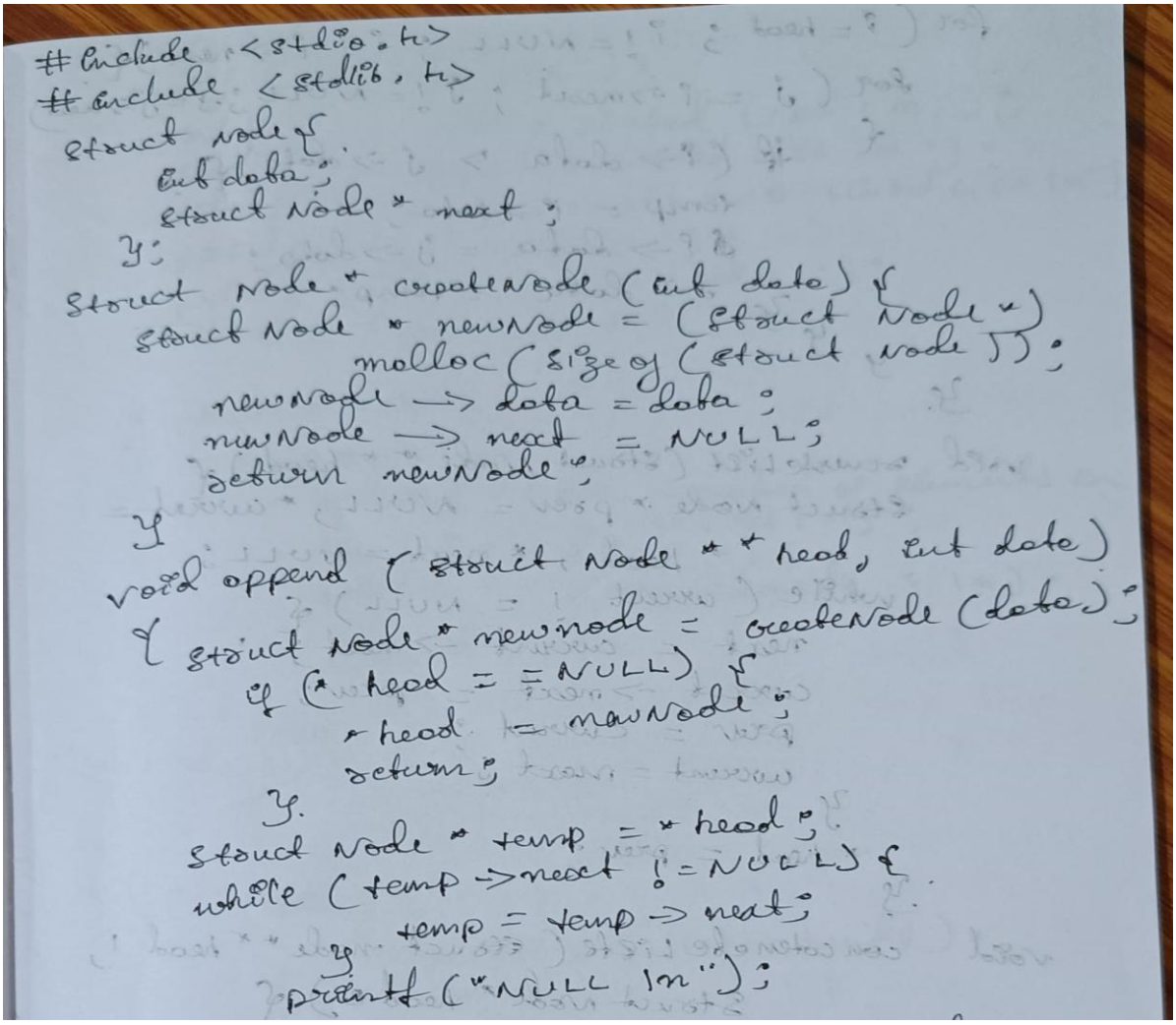
Problem 7:

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


week-8

6a) WAP to implement singly Link List with following operation.
Sort the linked list
Reverse the linked list
Concatenation of two linked lists

6b) WAP to implement single Link List to simulate Stack & queue operation

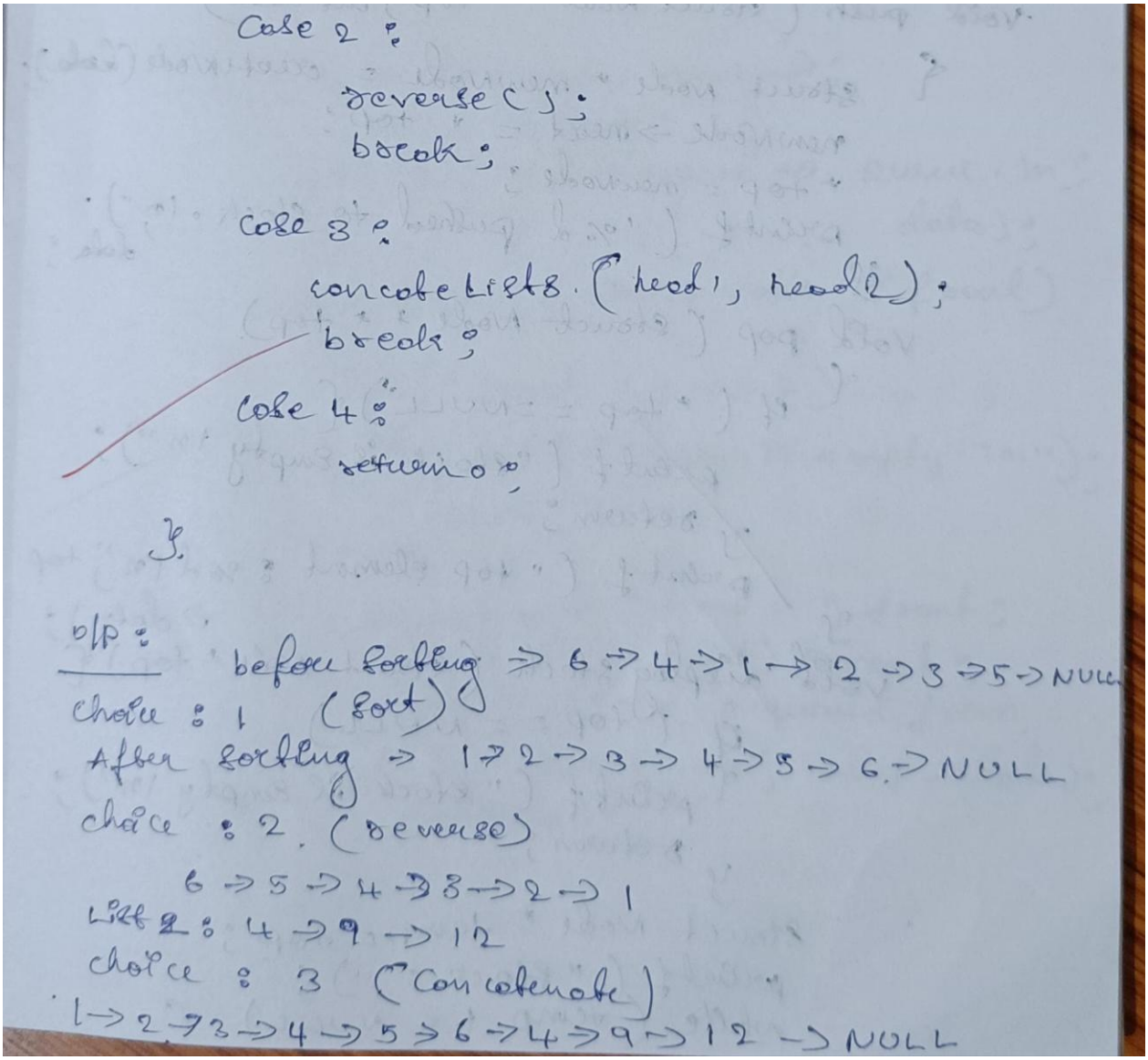

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct Node * next;
};
struct Node * createNode (int data) {
    struct Node * newNode = (struct Node *)
        malloc (sizeof (struct node));
    newNode -> data = data;
    newNode -> next = NULL;
    return newNode;
}
void append (struct Node ** head, int data)
{ struct Node * newnode = createNode (data);
    if (* head == NULL) {
        * head = newNode;
        return;
    }
    struct Node * temp = * head;
    while (temp -> next != NULL) {
        temp = temp -> next;
    }
    printf ("NULL \n");
```

```
3.
void sortlist (struct Node * head) {
printlist
    struct Node * temp = head;
    while (temp != NULL) {
      printf ("%d -> ", temp -> data);
      temp = temp -> next;
    }
    printf ("NULL \n");
}
void SortList (struct node * head) {
    if (head == NULL) 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) {
        temp = i -> data;
        i -> data = j -> data;
        j -> data = temp;
    }
}
}
void reverseList (struct Node ** head) {
    struct Node * prev = NULL, * current =
        * head, * next = NULL;
    while (current != NULL) {
        next = current -> next;
        current -> next = prev;
        prev = current;
        current = next;
    }
    * head = prev;
}
```

```
void concatenateLists(struct node **head1,
                      struct Node *head2){
    if(*head1 == NULL){
        *head1 = head2;
        return;
    }
    struct Node *temp = *head1;
    while(temp->next != NULL){
        temp = temp->next;
    }
    temp->next = head2;
}
int main(){
    struct Node *list1 = NULL;
    struct Node *list2 = NULL;
    int choice, data, n;
```

```
while(1){
    printf("\n menu:\n");
    printf("1. sort a Linked List\n");
    printf("2. Reverse a Lisked List\n");
    printf("3. concatenate two Linked List\n");
    printf("4. Exit\n");
    printf("Enter your choice");
    scanf("%d", &choice);

    switch(choice){
        case 1:
            printf("Enter number of elements in
                    the List:");
            scanf("%d", &n);
            printf("Enter elements:\n");
            SortList();
```



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;
}

void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = newNode;
}

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

void sortList(struct Node** head) {
    if (*head == NULL || (*head)->next == NULL)
        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) {
                temp = i->data;
                i->data = j->data;
                j->data = temp;
            }
        }
    }
}

void reverseList(struct Node** head) {
```

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

void concatenateLists(struct Node** head1, struct Node** head2) {
    if (*head1 == NULL) {
        *head1 = *head2;
        return;
    }
    struct Node* temp = *head1;
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = *head2;
}

int main() {
    struct Node* list1 = NULL;
    struct Node* list2 = NULL;

    insertAtEnd(&list1, 3);
    insertAtEnd(&list1, 1);
    insertAtEnd(&list1, 4);
    insertAtEnd(&list1, 2);

    insertAtEnd(&list2, 7);
    insertAtEnd(&list2, 6);
    insertAtEnd(&list2, 5);

    printf("List 1: ");
    printList(list1);
    printf("List 2: ");
    printList(list2);

    sortList(&list1);
```

```
    printf("Sorted List 1: ");
    printList(list1);

    reverseList(&list1);
    printf("Reversed List 1: ");
    printList(list1);

    concatenateLists(&list1, &list2);
    printf("Concatenated List: ");
    printList(list1);

    return 0;
}
```

Output:

```
PS C:\Users\Shashank U\Desktop\C> gcc .\queue.c
PS C:\Users\Shashank U\Desktop\C> .\a.exe
List 1: 3 -> 1 -> 4 -> 2 -> NULL
List 2: 7 -> 6 -> 5 -> NULL
Sorted List 1: 1 -> 2 -> 3 -> 4 -> NULL
Reversed List 1: 4 -> 3 -> 2 -> 1 -> NULL
Concatenated List: 4 -> 3 -> 2 -> 1 -> 7 -> 6 -> 5 -> NULL
PS C:\Users\Shashank U\Desktop\C>
```

Problem 8:
WAP to Implement Singly Linked List to simulate Stack Operations.

Code:

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

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

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

struct Node* top=NULL;

void push(int data){
    struct Node* newNode=createNode(data);
    newNode->next=top;
    top=newNode;
}

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

void peek(){
    if(top==NULL){
        printf("Empty stack!\n");
        return;
```

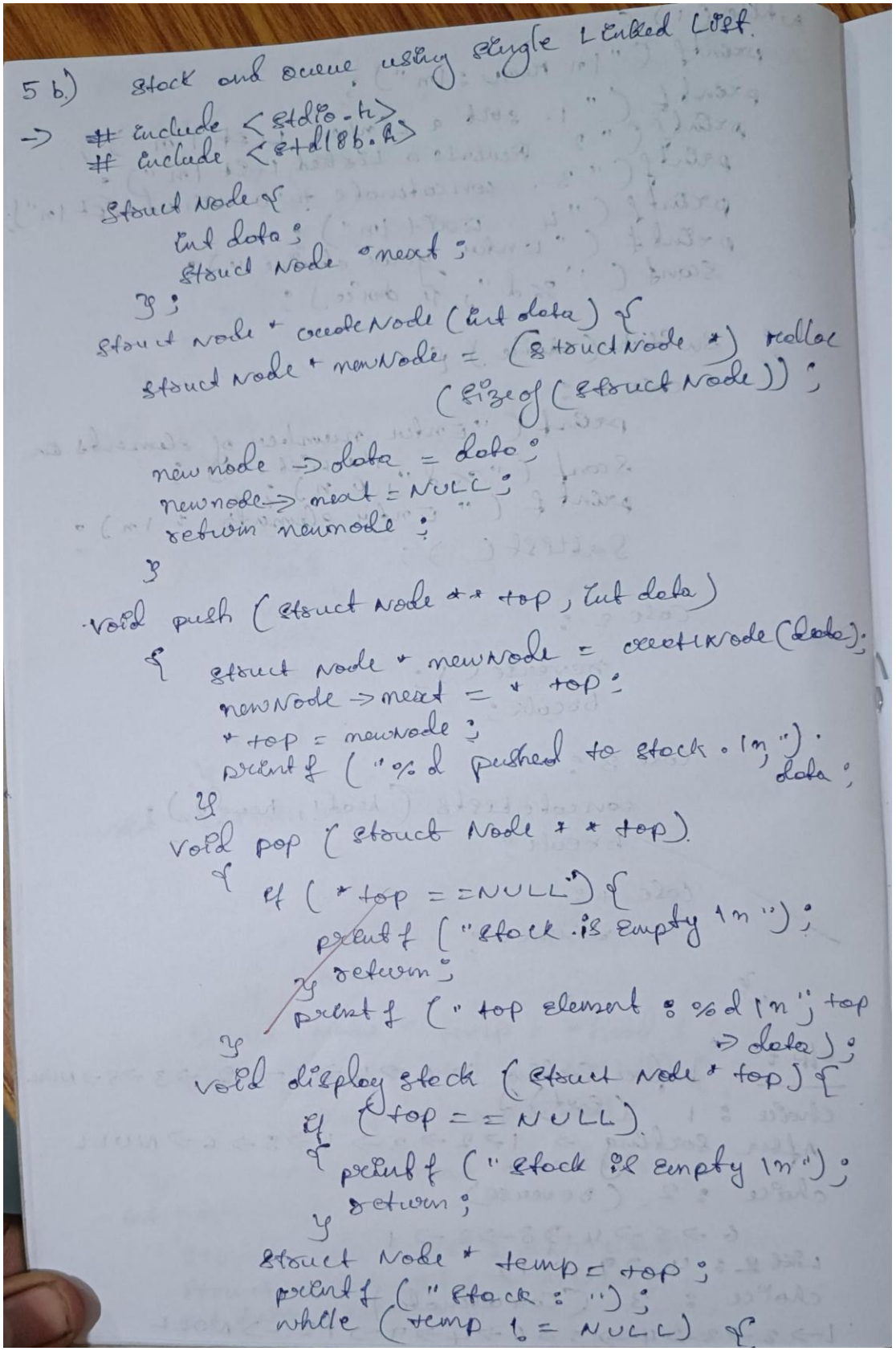
```
    }
    printf("Peek element:%d\n",top->data);
}

void display(){
    if(top==NULL){
        printf("Empty stack\n");
        return;
    }
    for(struct Node*temp=top;temp!=NULL;temp=temp->next){
        printf("%d,",temp->data);
    }
    printf("\n");
}

int main(){
    for(int i=15;i>0;i--){
        push(i);
    }
    pop();
    pop();
    display();
    return 0;
}
```

Output:

```
PS C:\Users\Shashank U\Desktop\C> gcc .\queue.c
PS C:\Users\Shashank U\Desktop\C> .\a.exe
3,4,5,6,7,8,9,10,11,12,13,14,15,
PS C:\Users\Shashank U\Desktop\C>
```

Problem 9:

WAP to Implement Single Link List to simulate Queue Operations.



```
5 b.) stack and queue using single Linked List.

-> #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 ("%d pushed to stack. \n", data);
   }
void pop (struct Node * * top)
   {
     if (*top == NULL) {
        printf ("stack is empty \n");
        return;
     printf ("top element : %d \n", top
                                -> data);
   }
void display stack (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 ("%d enqueued to queue. \n",
                                data);
    return;
  }
  (*rear) -> next = newNode;
  *rear = newNode;
  printf ("%d enqueued to queue. \n",
                                data);
}
void dequeue (struct Node ** front)
{
  if (*front == NULL)
  {
    printf ("Queue is empty \n");
    return;
  }
  struct Node * temp = *front;
  *front = (*front) -> next;
  printf ("%d dequeued from
          queue. \n", temp -> data);
  free (temp);
}
```

```
void peekqueue (struct Node *front) {
    if (front == NULL)
    {
        printf ("queue is empty \n");
        return;
    }
    printf ("front element : %d\n", front->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;
    do {
        printf ("\n choose an operation :\n");
        printf ("1. Stack : Push \n");
        printf ("2. Stack : Pop \n");
        printf ("3. Stack : Peek. \n");
        printf ("4. Stack : Display \n");
```

```
printf("5. Queue : Enqueue \n");
printf("6. Queue : Dequeue \n");
printf("7. Queue : Peek \n");
printf("8. Queue : Display \n");
printf("9. Exit \n");
printf("Enter your choice ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        printf("Enter value to push to Stack : ");
        scanf("%d", &data);
        push (&stackTop, data);
        break;

    case 2:
        pop (&stackTop);
        break;

    case 3:
        peek (stackTop);
        break;

    case 4:
        display Stack (stackTop);
        break;

    case 5:
        printf("Enter value to enqueue to
                                Queue : ");
        scanf("%d", &data);
        Enqueue (&queueFront, &queueRear,
                                data);
        break;

    case 6:
        dequeue (&queueFront);
        break;

    case 7:
        peekQueue (queueFront);
        break;
```

```
case 8:
    displayQueue(queueFront);
    break;
case 9:
    printf("Exiting the program.\n");
    break;
default:
    printf("Invalid choice. Try again.\n");
}
} while (choice != 9);
return 0;
}
```

o/p:

choice : 1. (push).
value : 23.
23 pushed to stack.
choice : 1
value : 34
34 pushed to stack.
choice : 4 (display).

34 -> 23 -> NULL.

choice : 3 (peek).

34

choice : 5 (enqueue)
value : 44
44 enqueued.
choice : 5 (enqueue)
value : 45
choice : 8.

44 -> 45 -> NULL.

choice : 6.

45 -> NULL.

choice : 7 (peek)

45.

Code:

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

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

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

struct Node *front=NULL,*rear=NULL;

void enqueue(int value){
   struct Node*newNode=createNode(value);
   if(front==NULL && rear==NULL){
      front=newNode;
      rear=newNode;
      return;
   }
   rear->next=newNode;
   rear=newNode;
}
void deque(){
   if(front==NULL && rear==NULL){
      printf("Queue is empty");
      return;
   }
   struct Node*temp=front;
   front=front->next;
   free(temp);
}
void front_(){
   if(front==NULL && rear==NULL){
      printf("Queue is empty");
      return;
   }
```

```
    printf("front element:%d\n",front->data);
}

void rear_(){
  if(front==NULL && rear==NULL){
      printf("Queue is empty");
      return;
    }
    printf("rear element:%d\n",rear->data);
}

void display(){
    if(front==NULL && rear==NULL){
      printf("Queue is empty");
      return;
    }
    for(struct Node*temp=front;temp!=NULL;temp=temp->next){
      printf("%d,",temp->data);
    }
    printf("\n");
}

int main(){
    for(int i=1;i<16;i++){
      enqueue(i);
    }
    deque();
    deque();
    front_();
    rear_();
    display();
    return 0;
}
```

Output:

```
PS C:\Users\Shashank U\Desktop\C> gcc .\queue.c
PS C:\Users\Shashank U\Desktop\C> .\a.exe
front element:3
rear element:15
3,4,5,6,7,8,9,10,11,12,13,14,15,
PS C:\Users\Shashank U\Desktop\C>
```

Problem 10:

WAP to Implement doubly link list with primitive operations

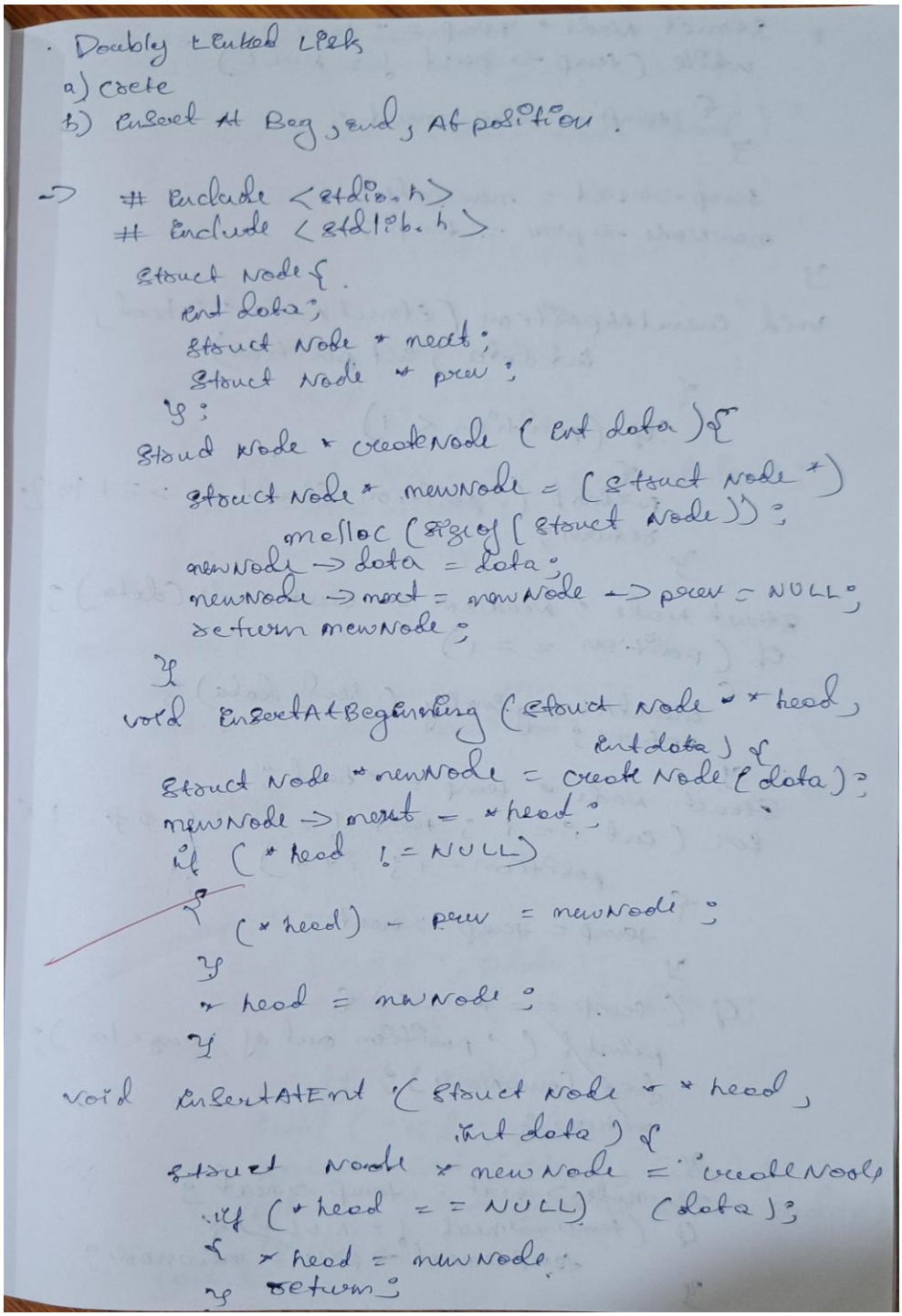
a) Create a doubly linked list.

b) Insert a new node at the beginning.

c) Insert the node based on a specific location

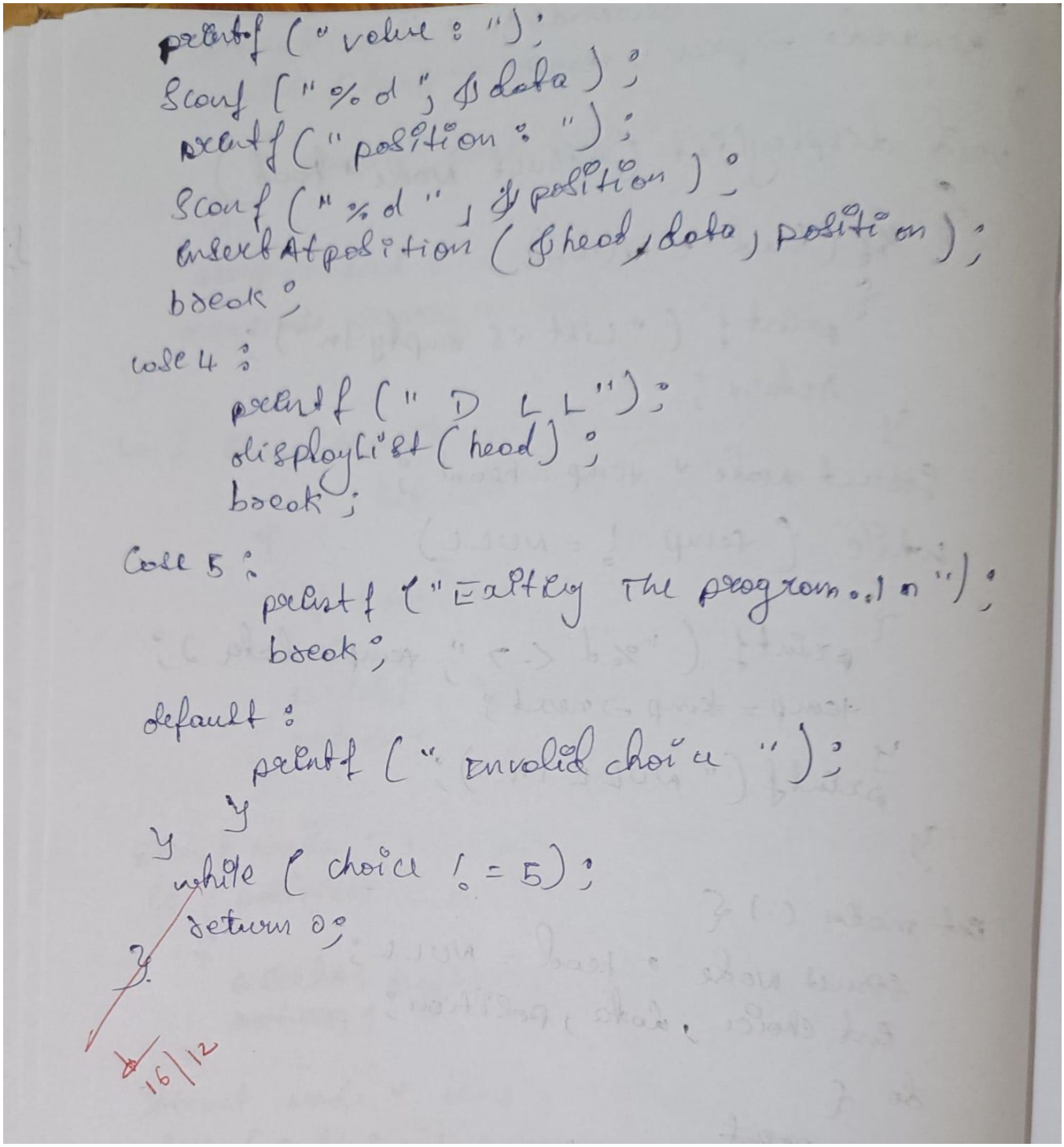
d) Insert a new node at the end.

e) Display the contents of the list



```
struct Node * temp = * head ;
while (temp -> next != NULL)
  { temp = temp ->next ;
  }
temp ->next = newNode ;
newNode -> prev = temp ;
}
void insertAtposition (struct Node * * head,
                       int data , int position)
  {
    if (position < 1)
    {
    printf ("position should be >= 1 \n");
    return ;
    }
struct Node * NewNode = createNode (data) ;
if (position == 1)
  { insertAtBeginning ( head, data) ;
    return ;
  }
struct Node * temp = * head ;
for ( int i = 1 ; temp != NULL && i <
        position - 1 ; i++ )
  {
    temp = temp -> next ;
  }
if (temp == NULL) {
    printf (" position out of range \n");
    free (newNode) ;
    return ;
  }
newnode -> next = temp -> next ;
if (temp -> next != NULL) {
      temp -> next -> prev = newnode ;
}
```

```
    temp -> next = newNode;
    newNode -> prev = temp;
}
void displayList (struct node * head)
{
    if (head == NULL).
    {
        printf ("List is empty \n");
        return;
    }
    struct node * temp = head;
    while (temp != NULL)
    {
        printf ("%d <-> ", temp -> data);
        temp = temp -> next;
    }
    printf ("NULL \n");
}
int main () {
    struct node * head = NULL;
    int choice, data, position;
    do {
        ~~printf~~
        switch (choice) {
        case 1:
        printf ("value")
        scanf ("%d", &data);
        InsertAtBegnry (&head, data);
        case 2:
            printf ("value");
            scanf ("%d", &data);
            InsertAtEnd (&head, data);
            printf ("Node inserted at the end \n");
            break;
        case 3;
```



Code:

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

struct Node{
```

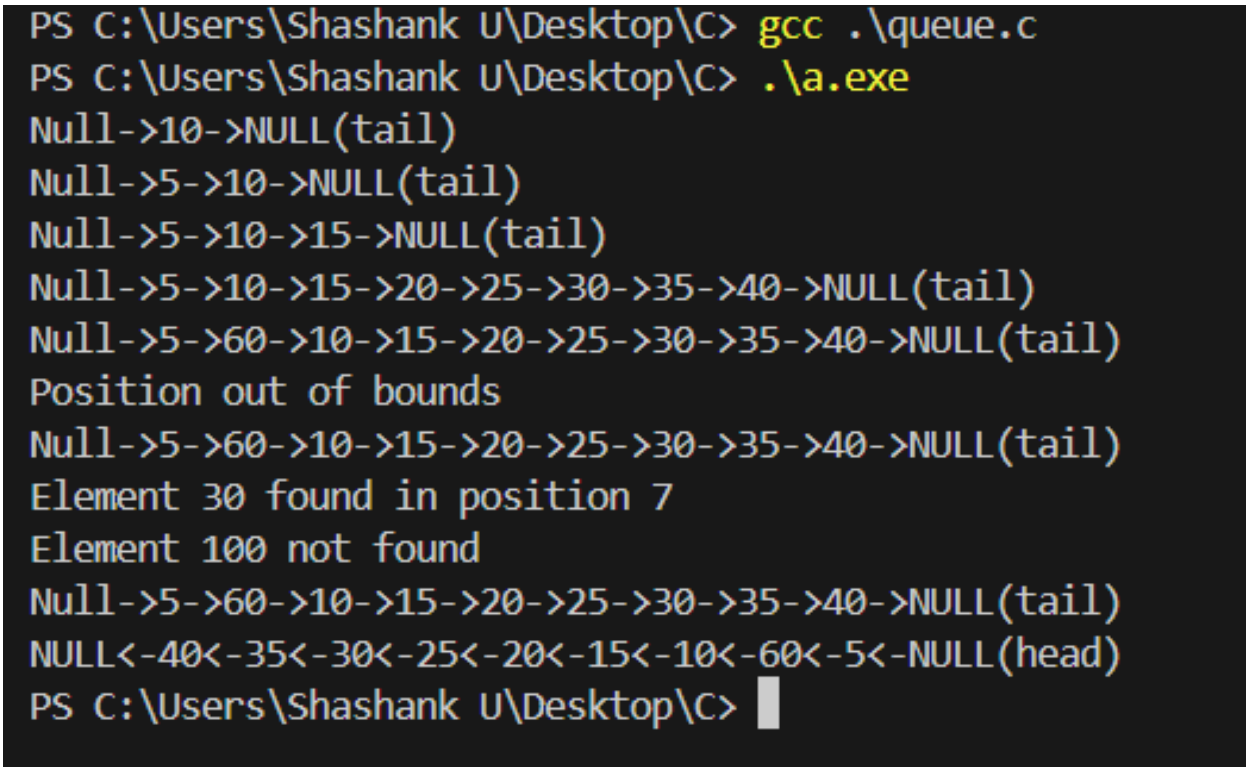
```
    int data;
    struct Node*prev;
    struct Node*next;
};

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

void insert_at_begin(struct Node**head,struct Node**tail,int value){
    struct Node*node=createNode(value);
    if(*head==NULL){
        *head=*tail=node;
        return;
    }
    node->next=*head;
    (*head)->prev=node;
    *head=node;
}

void insert_at_end(struct Node**head,struct Node**tail,int value){
    struct Node*node=createNode(value);
    if(*head==NULL){
        *head=*tail=node;
        return;
    }
    (*tail)->next=node;
    node->prev=*tail;
    *tail=node;
}

void insert_at_pos(struct Node**head,struct Node**tail,int pos,int value){
    struct Node*node=createNode(value);
    if(*head==NULL || pos==1){
        insert_at_begin(head,tail,value);
        return;
    }
    if(pos<=0){
```

```
        printf("Invalid position\n");
        return;
    }
    struct Node* temp=*head;
    for(int i=1;i<pos-1;i++){
        if(temp==NULL){
            printf("Position out of bounds\n");
            free(node);
            return;
        }
        temp=temp->next;
    }

    if(temp->next==NULL){
        insert_at_end(head,tail,value);
        return;
    }
    node->next=temp->next;
    node->prev=temp;
    temp->next->prev=node;
    temp->next=node;
}

void search(struct Node**head,int ele){
    struct Node*temp=*head;
    int i=1;
    if(*head==NULL){
        printf("linked list is empty\n");
        return;
    }
    while(temp!=NULL){
        if(temp->data==ele){
            printf("Element %d found in position %d\n",ele,i);
            return;
        }
        i++;
        temp=temp->next;
    }
    printf("Element %d not found\n",ele);
}

void display(struct Node**head){
```

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

void reverse_print(struct Node**tail){
    if(*tail==NULL){
        printf("linked list is empty\n");
        return;
    }
    struct Node*temp=*tail;
    printf("NULL<-");
    while(temp!=NULL){
        printf("%d<-",temp->data);
        temp=temp->prev;
    }
    printf("NULL(head)\n");
}

int main(){
    struct Node *head=NULL,*tail=NULL;
    insert_at_begin(&head,&tail,10);
    display(&head);
    insert_at_begin(&head,&tail,5);
    display(&head);
    insert_at_end(&head,&tail,15);
    display(&head);
    insert_at_end(&head,&tail,20);
    insert_at_end(&head,&tail,25);
    insert_at_end(&head,&tail,30);
    insert_at_end(&head,&tail,35);
    insert_at_end(&head,&tail,40);
    display(&head);
    insert_at_pos(&head,&tail,2,60);
```

```
    display(&head);
    insert_at_pos(&head,&tail,80,6);
    display(&head);
    search(&head,30);
    search(&head,100);
    display(&head);
    reverse_print(&tail);
    return 0;
}
```

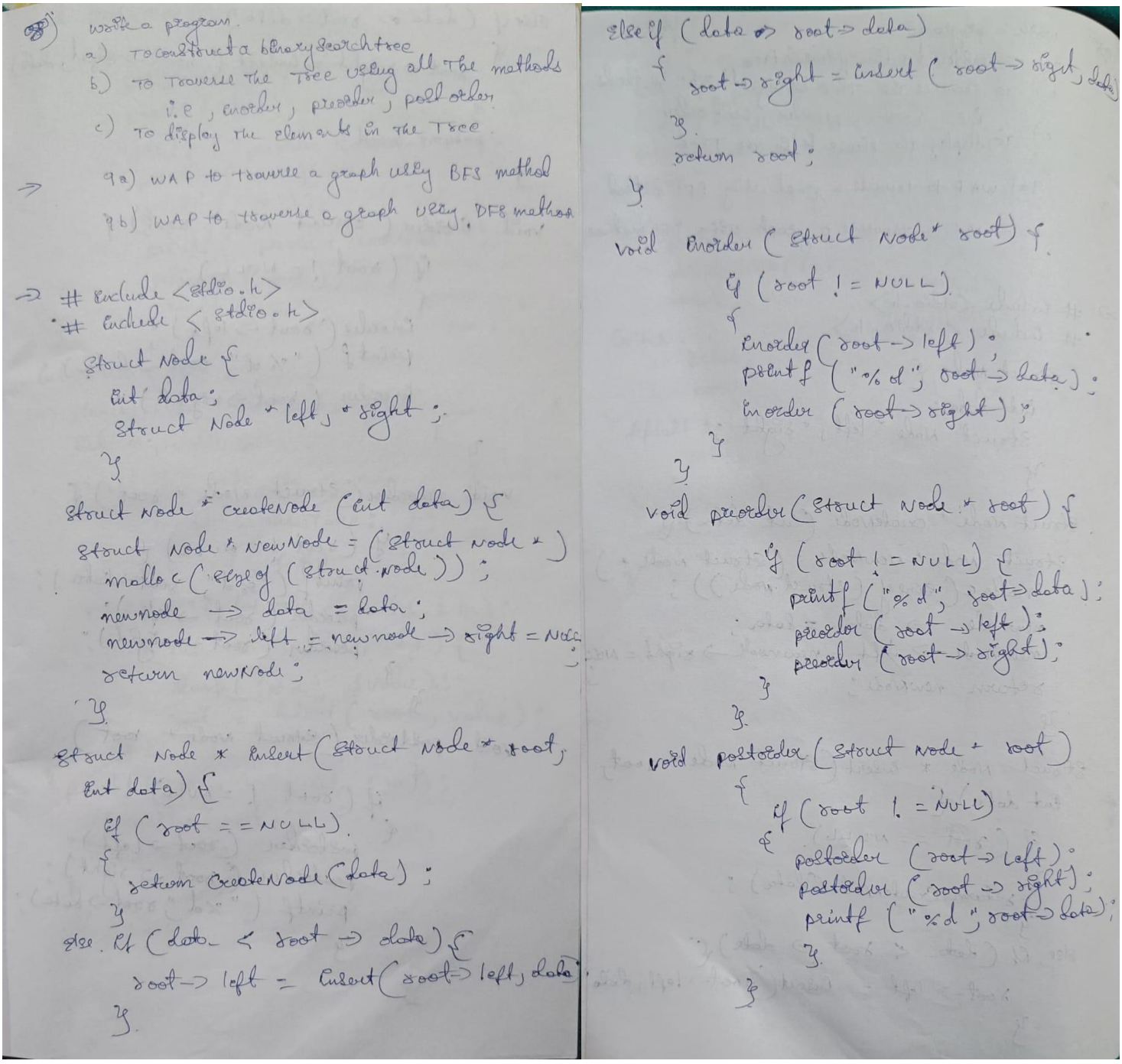
Output:



Problem 11: 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 traversal order



```
void displayTree ( struct Node * root ) {
    printf ( "Inorder Traversal" ):
     inorder (root);
     printf ("\n");
      printf ( "preorder Traversal : ");
      preorder ( root );
      printf ("\n");
      printf ( "preorder Traversal: ");
       postorder ( root );
       printf ("\n");
}
int main ( ){
   struct Node * root = NULL;
   int n, value;
   printf ( " Enter the number of element to
          insert in the Tree : ");
     Scanf ("%d", &n);
   printf ( " Enter the elements : \n");
    for ( int i = 0; i < n; i++ ) {
          Scanf ( "%d", &value);
          root = Insert ( root, value);
    }
printf ( " Tree Traversal : \n");
  displayTree ( root);
  return 0;
}
```

o/p

Enter the nor of nodes you wat to enter : 5

1 : 50
2 : 30
3 : 70
4 : 20
5 : 40

In order traversal : 20 30 40 50 70
pre order traversal : 50 30 20 40 70
post order traversal : 20 40 30 70 50

Code:

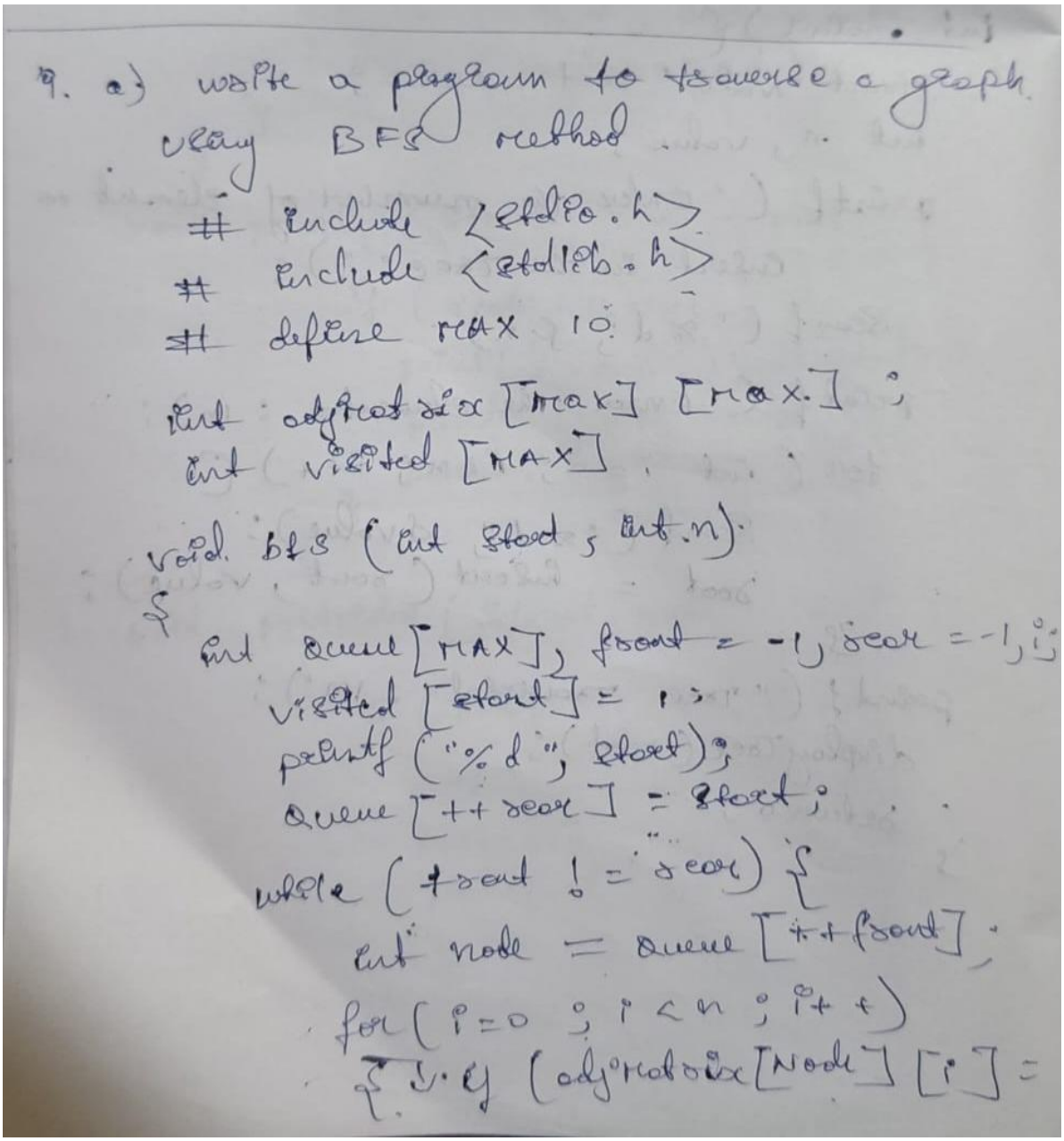
```
#include<stdio.h>
#include<stdlib.h>

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

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

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

    if(temp->data>root->data){
        if(root->right!=NULL){
            insert(root->right,temp);
        }
        else{
            root->right=temp;
        }
    }
}

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

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

    do{
        struct Node*temp=createNode();
        if(root==NULL){
            root=temp;
        }
        else{
            insert(root,temp);
        }
        printf("Do you want to continue? if Yes the enter Y/y:");
        scanf(" %c",&ch);
    }while(ch=='Y' || ch=='y');

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

```
    printf("Inorder traversal:");
    inorder(root);
    printf("\n");
    printf("Postorder traversal:");
    postorder(root);
    printf("\n");
    return 0;
}
```

```
Enter value:20
Do you want to continue? if Yes the enter Y/y:y
Enter value:15
Do you want to continue? if Yes the enter Y/y:y
Enter value:30
Do you want to continue? if Yes the enter Y/y:y
Enter value:50
Do you want to continue? if Yes the enter Y/y:y
Enter value:60
Do you want to continue? if Yes the enter Y/y:y
Enter value:45
Do you want to continue? if Yes the enter Y/y:y
Enter value:25
Do you want to continue? if Yes the enter Y/y:y
Enter value:36
Do you want to continue? if Yes the enter Y/y:y
Enter value:78
Do you want to continue? if Yes the enter Y/y:y
Enter value:88
Do you want to continue? if Yes the enter Y/y:n
Preorder traversal:20 15 30 25 50 45 36 60 78 88
Inorder traversal:15 20 25 30 36 45 50 60 78 88
Postorder traversal:15 25 36 45 88 78 60 50 30 20
BFS traversal:20 15 30 25 50 45 60 36 78 88
```

Problem 12: a) Write a program to traverse a graph using BFS method.
b) Write a program to traverse a graph using DFS method.



```
9. a) write a program to traverse a graph
   using BFS method.

#include <stdio.h>
#include <stdlib.h>
#define MAX 10

int adjmatrix[MAX][MAX];
int visited[MAX];

void bfs(int start, int n)
{
    int queue[MAX], front = -1, rear = -1, i;
    visited[start] = 1;
    printf("%d ", start);
    queue[++rear] = start;
    while (front != rear) {
        int node = queue[++front];
        for (i=0; i<n; i++)
            if (adjmatrix[node][i] =
```

```
    if (!visited[i])
    {
        printf("%d", i);
        visited[i] = 1;
        queue[++rear] = i;
    }
  }
}

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

    printf("Enter the adjacency matrix:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &adjmatrix[i][j]);
        }
    }

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

    printf("Enter the starting vertex:");
    scanf("%d", &start);

    printf("Breadth First Traversal starting
        from vertex %d:", start);
    bfs(start, n);
    return 0;
}
```

o/p

Enter the number of vertices : 3
Enter the adjacency matrix

0 1
1 2
2 3
3 4
4 5

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

```
-> #include <stdio.h>
   #include <stdlib.h>
   #define MAX_vertices 1000
   typedef struct graph {
       int adj[MAX_vertices][MAX_vertices];
       int vertices;
   } graph;
   void DFS(graph *g, int vertex, int
        , visited) {
       visited[vertex] = 1;
   for (int i=0; i<g->vertices; i++)
   {
       if (g->adj[vertex][i] == 1 && !
           visited[i]) {
```

```
        DFS(g, i, visited);
    }
}
}
int isConnected(graph *g) {
    int visited[MAX_VERTICES] = {0};
    DFS(g, 0, visited);
    for (int i = 0; i < g->vertices; i++) {
        if (!visited[i]) {
            return 0;
        }
    }
    return 1;
}
int main() {
    graph g;
    int edges, u, v;
    printf("Enter the no of vertices");
    scanf("%d", &g.vertices);
    for (int i = 0; i < g.vertices; i++)
    {
        for (int j = 0; j < g.vertices;
                                j++) {
            g.adj[i][j] = 0;
        }
    }
    printf("Enter The no of Edges: ");
    scanf("%d", &Edges);
    printf("Enter The Edges (u, v) where
```

u & v are ~~vertices~~ vertex indices : \n");

```
for (int i=0; i< edges ; i++)
{
    scanf("%d %d", &u, &v);
    g.adj[u][v] = 1;
    g.adj[v][u] = 1;
}
if (isconnected(&g)){
    printf("the graph is connected \n");
}
else{
    printf("the graph is not connected : \n");
}
return 0;
}
```

o/p

Enter the number of vertices : 5
Enter the number of edges : 4
Enter the Edges (u,v) where u & v are vertex indices :

0 1
1 2
2 3
3 4

the graph is connected.

Code:

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

void BFS(int adj[15][15],int V,int i_vertex){
    int q[5],front=0,rear=0;
    bool visited[5]={false};
    visited[i_vertex]=true;
    q[rear++]=i_vertex;
    while(front<rear){
        int curr=q[front++];
        printf("%d ",curr);
        for(int i=0;i<V;i++){
            if(adj[curr][i]==1 && !visited[i]){
                visited[i]=true;
                q[rear++]=i;
            }
        }
    }
}

void addEdge(int adj[15][15],int row,int column){
    adj[row][column]=adj[column][row]=1;
}

void DFS(int adj[15][15],int V,int i_vertex,bool visited[V]){
    visited[i_vertex]=true;
    printf("%d ",i_vertex);
    for(int i=0;i<V;i++){
        if(adj[i_vertex][i]==1 && !visited[i]){
            DFS(adj,V,i,visited);
        }
    }
}

int main(){
    int V=5;
    int adj[15][15]={0};
    addEdge(adj,0,1);
    addEdge(adj,0,2);
```

```
    addEdge(adj,1,3);
    addEdge(adj,1,4);
    addEdge(adj,2,4);
    addEdge(adj,3,4);
    printf("DFS Traversal with initial node 0:");
    bool visited[5]={false};
    DFS(adj,V,0,visited);
    printf("\nBFS Traversal with initial node 0:");
    BFS(adj,V,0);
    return 0;
}
```