VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Data Structures using C Lab

(23CS3PCDST)

Submitted by

Shaikh Uzair Ahmed (1BM23CS307)

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 **Shaikh Uzair Ahmed (1BM23CS307),** 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 academicrequirements in respect of Data Structures using C Lab (23CS3PCDST) work prescribed for the said degree.

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

Index

Github Link:
https://github.com/Shaikh-Uzair-Ahmed/1BM23CS307

Program 1

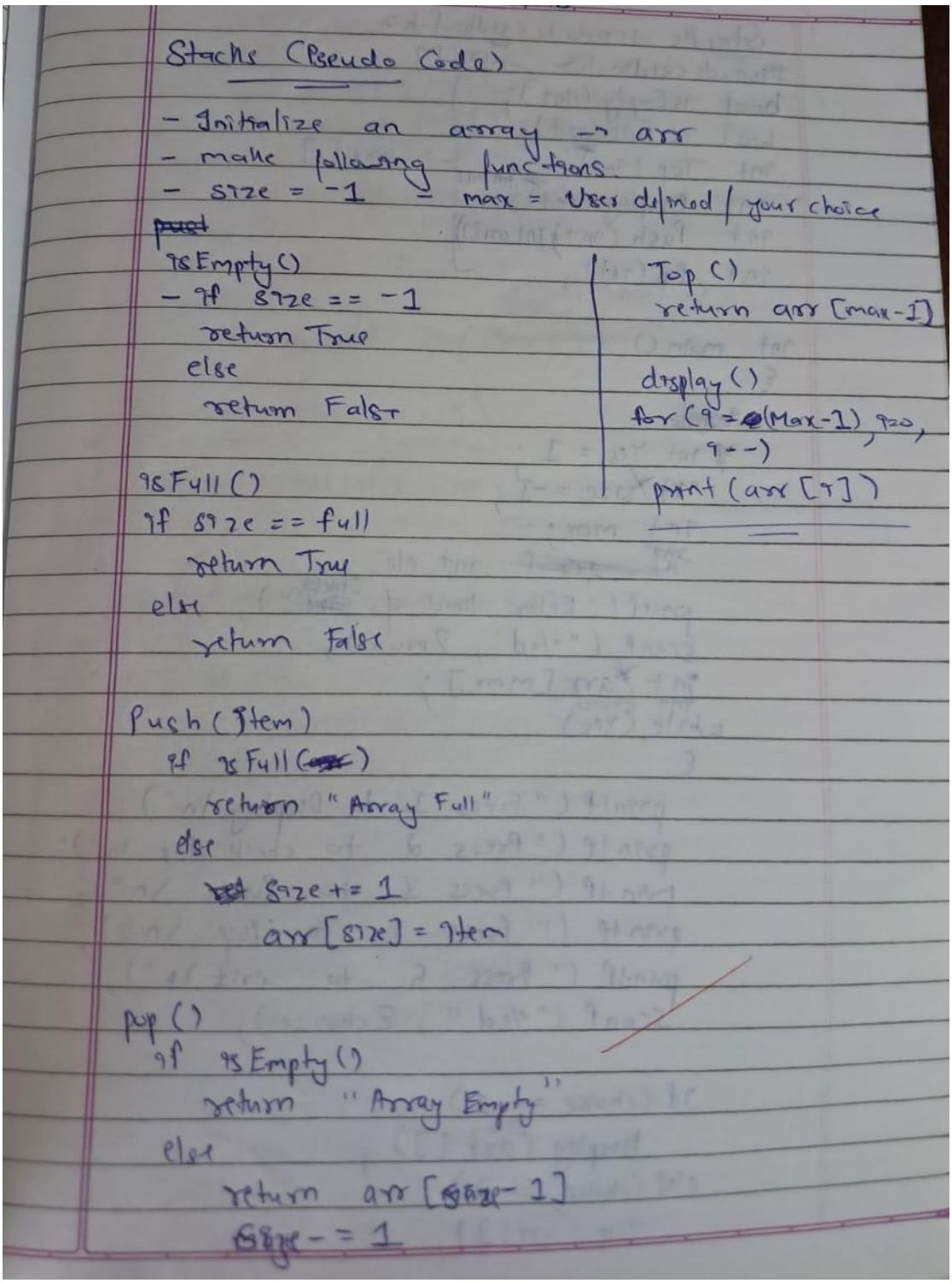
Write a program to simulate the working of stack using an array with the following:

a) Push

b) Pop

c) Display

The program should print appropriate messages for stack overflow, stack underflow

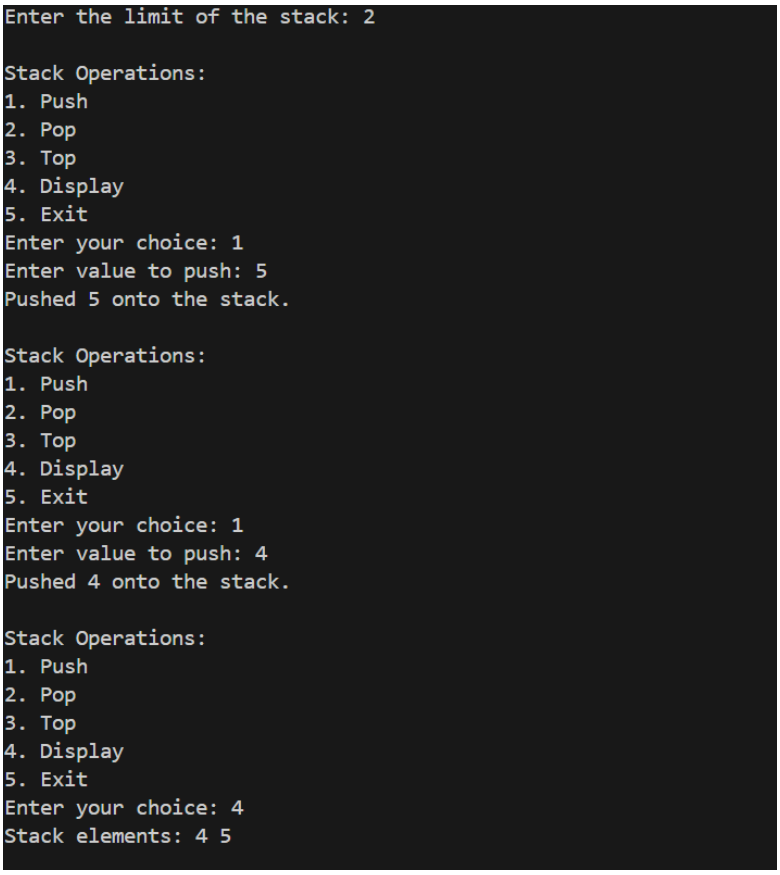


Code:

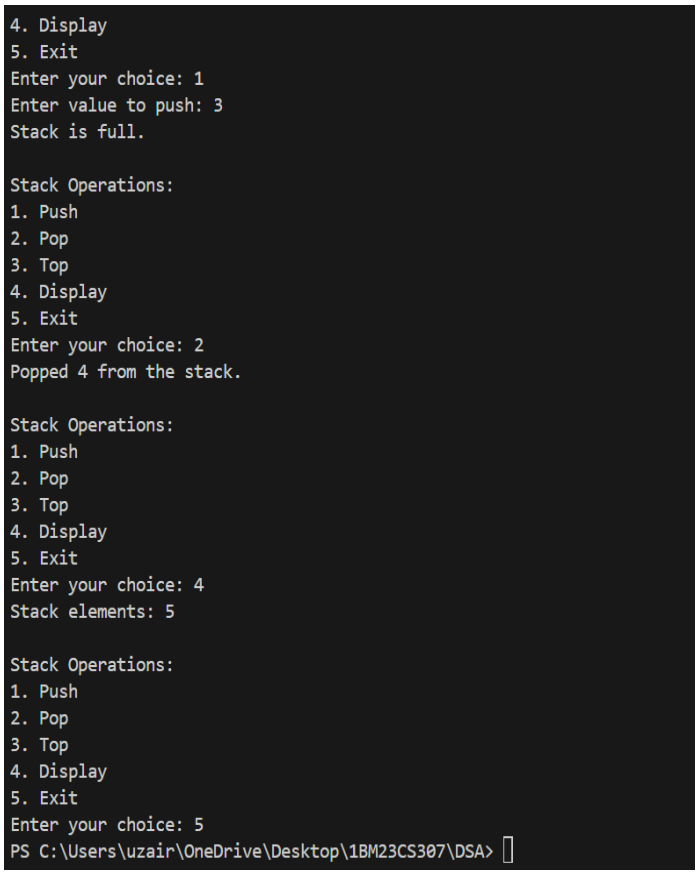
```
#include <stdio.h>
#include <stdlib.h>

int top = -1;


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

int isFull(int arr[], int limit) {
   return top == limit - 1;
}

int Top(int arr[]) {
   if (isEmpty(arr)) {
      printf("Stack is empty.\n");
      return -1;
   }
   return arr[top];
}

void Display(int arr[]) {
   if (isEmpty(arr)) {
      printf("Stack is empty.\n");
      return;
   }
   printf("Stack elements: ");
   for (int i = top; i >= 0; i--) {
      printf("%d ", arr[i]);
   }
   printf("\n");
}


void Push(int value, int arr[], int limit) {
   if (isFull(arr, limit)) {
      printf("Stack is full.\n");
   } else {
      top++;
      arr[top] = value;
      printf("Pushed %d onto the stack.\n", value);
   }
}
```

```
void Pop(int arr[]) {
    if (isEmpty(arr)) {
        printf("Stack is empty.\n");
    } else {
        printf("Popped %d from the stack.\n", arr[top]);
        top--;
    }
}

int main() {
    int limit;

    printf("Enter the limit of the stack: ");
    scanf("%d", &limit);

    int arr[limit];

    while (1) {
        int choice, value;
        printf("\nStack Operations:\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Top\n");
        printf("4. Display\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                Push(value, arr, limit);
                break;
            case 2:
                Pop(arr);
                break;
            case 3:
                printf("Top element is: %d\n", Top(arr));
                break;
            case 4:
                Display(arr);
                break;
            case 5:
                exit(0);
            default:
```

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

    return 0;
}
```



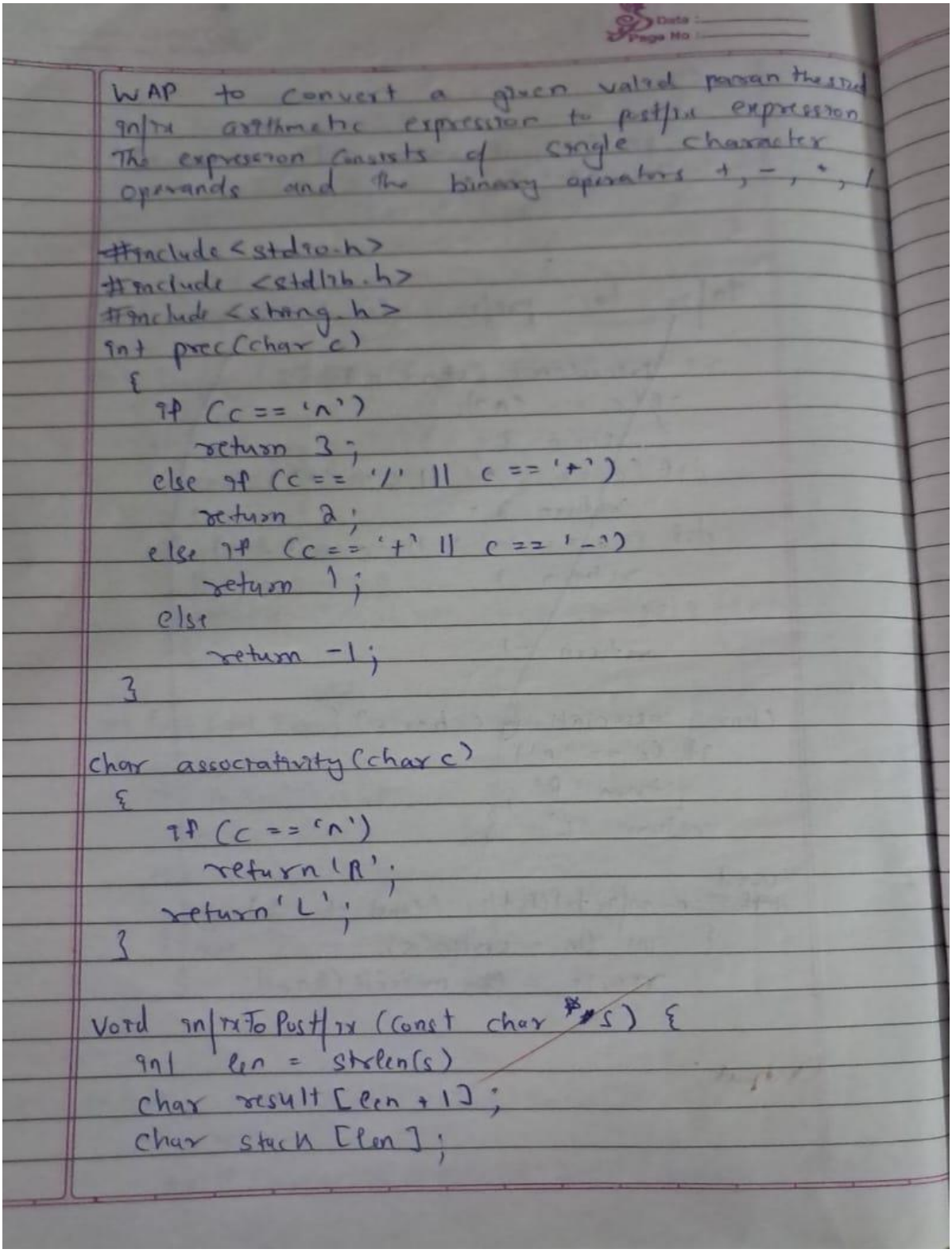
Output 1



Output 2

Program 2

Write a program to convert a given valid parenthesized in-fix arithmetic expression to post-fix expression.The expression consists of single character operands and the binary operators +,-,*,/.



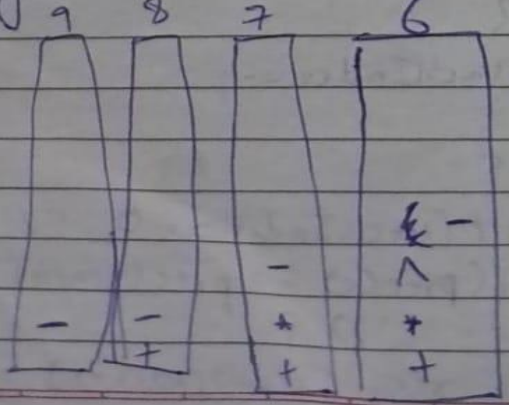
```
WAP to convert a given valid paran thesized
infix arithmetic expression to postfix expression
The expression consists of single character
operands and the binary operators +, -, *, /

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

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

void infixToPostfix (const char *s) {
   int len = strlen(s)
   char result [len + 1];
   char stack [len];
```

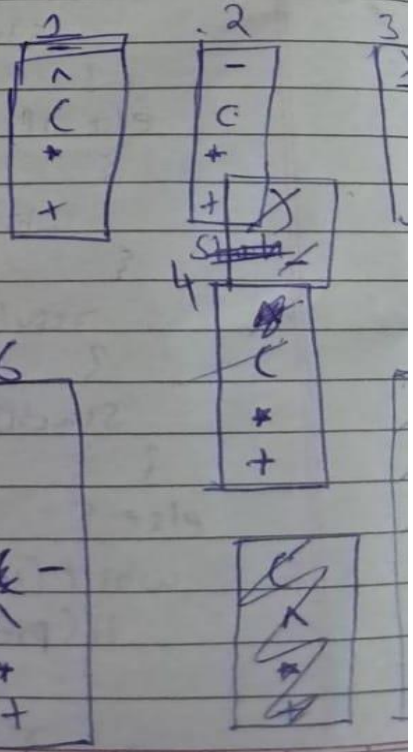
```
int resultIndex = 0;
int stackIndex = -1;

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[resultIndex++] = c;
    }
    else if (c == '(') {
        stack[++stackIndex] = c;
    }
    else if (c == ')')
    {
        while(stackIndex >= 0 && stack[stackIndex] != '(')
        {
            result[resultIndex++] = stack[stackIndex--];
        }
        stackIndex--;
    }
    else {
    while (stackIndex >= 0 && (prec(c) < prec(stack[stackIn
        || (prec(c) == prec(stack[stackIndex]) && associativity(c)
        == 'L')) {
            result[resultIndex++] = stack[stackIndex--];
    }
```

```
stack[++stackIndex] = c;
}//end for //end else
}// end function  // end for

while (stackIndex >= 0){
    result[resultIndex++] = stack[stackIndex--];
}

result[resultIndex] = '\0';
printf("%s\n", result);

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

Output:

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





Code:

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

int prec(char c){
    if (c == '+' || c == '-') {
        return 1;
    }
    if (c == '*' || c == '/') {
        return 2;
    }
    if (c == '^') {
        return 3;
    }
    return 0;
}

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

void infixtopostfix(const char *s){
    int len = strlen(s);
    char result[len+1];
    char stack[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'))
        {
            result[resultIndex++] = c;
        }
        else if(c=='('){
            stack[++StackIndex]=c;
        }
```

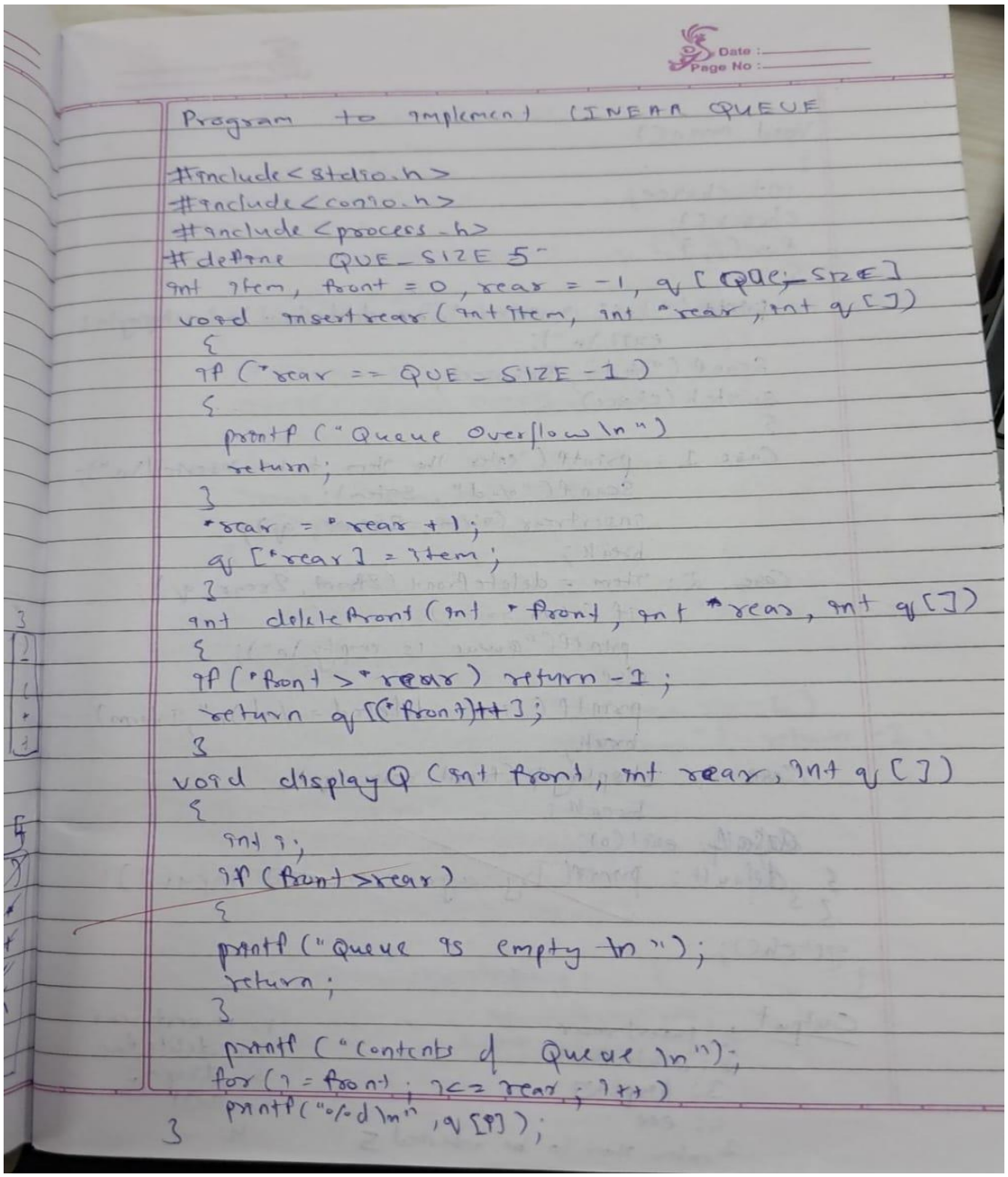
```
        else if(c==')'){
            while (StackIndex>=0 && stack[StackIndex]!='(')
            {
                result[resultIndex++]=stack[StackIndex--];
            }
           StackIndex--;
        }
        else{
            while (StackIndex>=0 && (prec(c))<prec(stack[StackIndex]) ||
(prec(c)==prec(stack[StackIndex]) && associativity(c)=='L'))
            {
                result[resultIndex++]=stack[StackIndex--];
            }
            stack[++StackIndex]=c;
        }
    }

    while (StackIndex>=0)
    {
        result[resultIndex++]=stack[StackIndex--];
    }
    result[resultIndex++]='\0';
    printf("%s\n",result);
}

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

```
PS C:\Users\uzair\OneDrive\Desktop\1BM23CS307\DSA> cd "c:\Users\uzair\OneDrive\Desktop\1BM23CS307\DSA\" ; if ($?) { gcc infi
xpostfix.c -o infixpostfix } ; if ($?) { .\infixpostfix }
abcd^e-fgh*+^*+i-
```

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.



```
Program to implement LINEAR QUEUE

#include<stdio.h>
#include<conio.h>
#include<process.h>
#define QUE_SIZE 5
int item, front = 0, rear = -1, q[QUE_SIZE]
void insertrear(int item, int *rear, int q[])
{
if(*rear == QUE_SIZE-1)
{
printf("Queue Overflow\n")
return;
}
*rear = *rear + 1;
q[*rear] = item;
}
int deletefront(int *front, int *rear, int q[])
{
if(*front > *rear) return -1;
return q[(*front)++];
}
void displayQ(int front, int rear, int q[])
{
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;
 clrscr();
 for(;;)
 {
  printf("\n1: insertrear\n2: deletefront\n3: display\n
        exit\n");
  scanf("%d", &choice);
  switch(choice)
  {
  case 1: printf("enter the item to be inserted\n");
          scanf("%d", &item);
          insertrear(item, &rear, q);
          break;
  case 2: item = deletefront(&front, &rear, q);
          if(item == -1)
           printf("queue is empty\n");
          else
          printf("item deleted = %d\n", item);
          break;
  case 3: displayQ(front, rear, q);
          break;
  default: exit(0);
  }
 default: printf("Try again invalid input");
 }
getch();
}
```

Output

1. insertrear
2: deletefront
3: display
4: exit
1 enter item to be inserted 5

1. insertrear
2. deletefron
3. display
4: exit
4

Code:

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

#define QUE_SIZE 5

int item, front = 0, rear = -1, q[N];

void insertrear(int item, int *rear, int q[]) {
   if (*rear == N - 1) {
      printf("Queue Overflow \n");
      return;
   }
   *rear = *rear + 1;
   q[*rear] = item;
}

int Deque(int *front, int *rear, int q[]) {
   if (*front > *rear) {
      return -1;
   }
   return q[(*front)++];
}

void displayQ(int front, int rear, int q[]) {
   if (front > rear) {
      printf("Queue is empty \n");
      return;
   }
   printf("Contents of Queue:\n");
   for (int i = front; i <= rear; i++) {
      printf("%d \n", q[i]);
   }
}

int main() {
   int choice;
   for (;;) {
      printf("\n1: Insert rear\n2: Delete front\n3: Display\n4: Exit\n");
      scanf("%d", &choice);
```

```
        switch (choice) {
            case 1:
                printf("Enter the item to be inserted: ");
                scanf("%d", &item);
                insertrear(item, &rear, q);
                break;
            case 2:
                item = Deque(&front, &rear, q);
                if (item == -1)
                    printf("Queue is empty\n");
                else
                    printf("Item deleted = %d \n", item);
                break;
            case 3:
                displayQ(front, rear, q);
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice, please try again.\n");
        }
    }
    return 0;
}
```

```
1: Insert rear
2: Delete front
3: Display
4: Exit
1
Enter the item to be inserted: 10

1: Insert rear
2: Delete front
3: Display
4: Exit
2
Item deleted = 2

1: Insert rear
2: Delete front
3: Display
4: Exit
2
Item deleted = 5

1: Insert rear
2: Delete front
3: Display
4: Exit
3
Contents of Queue:
2
4
10
```

Output 1

```
1: Insert rear
2: Delete front
3: Display
4: Exit
1
Enter the item to be inserted: 2

1: Insert rear
2: Delete front
3: Display
4: Exit
1
Enter the item to be inserted: 5

1: Insert rear
2: Delete front
3: Display
4: Exit
1
Enter the item to be inserted: 2

1: Insert rear
2: Delete front
3: Display
4: Exit
1
Enter the item to be inserted: 4
```

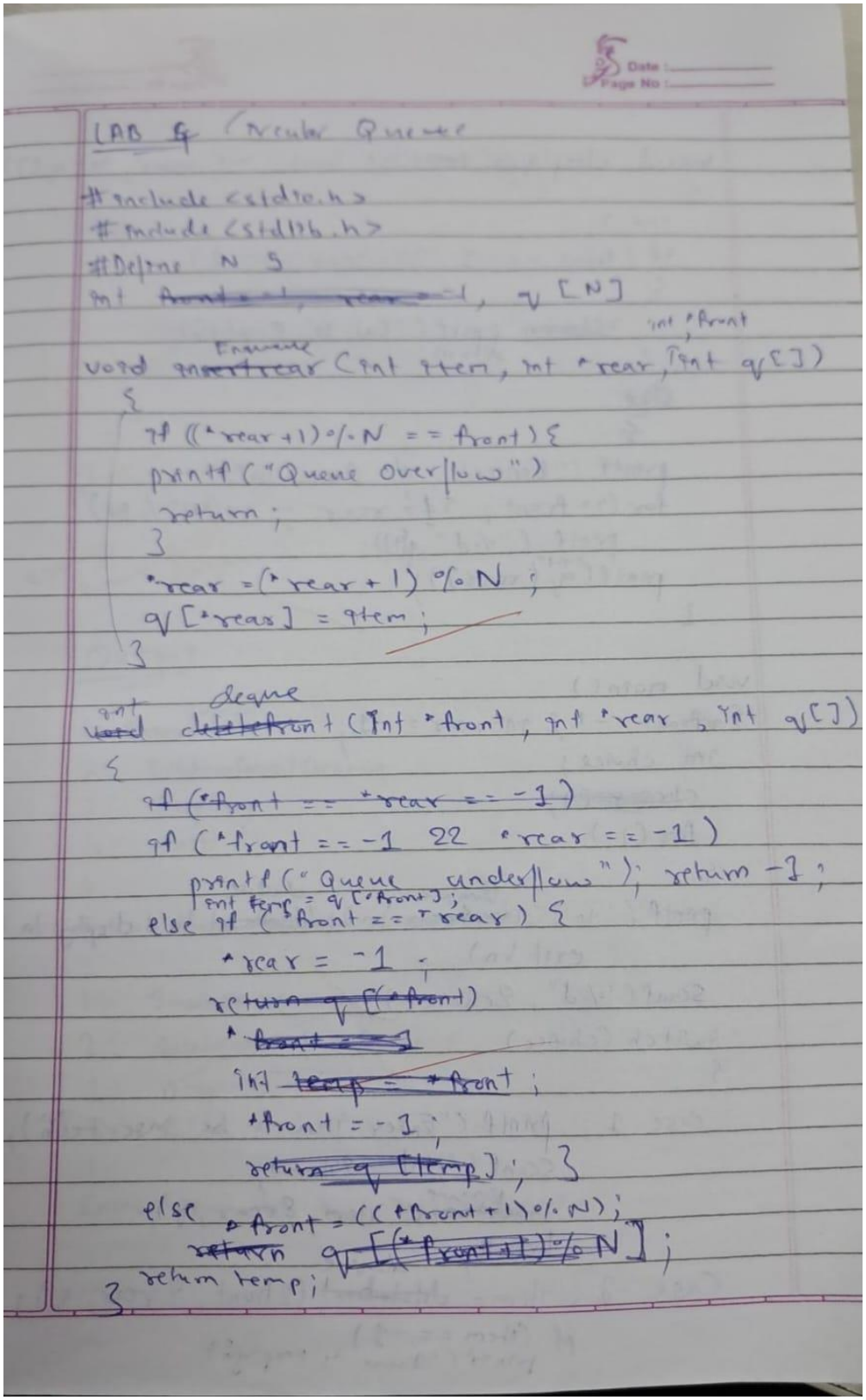
Output 2

```
1: Insert rear
2: Delete front
3: Display
4: Exit
4
PS C:\Users\uzair\OneDrive\Desktop\1BM23CS307\DSA>
```

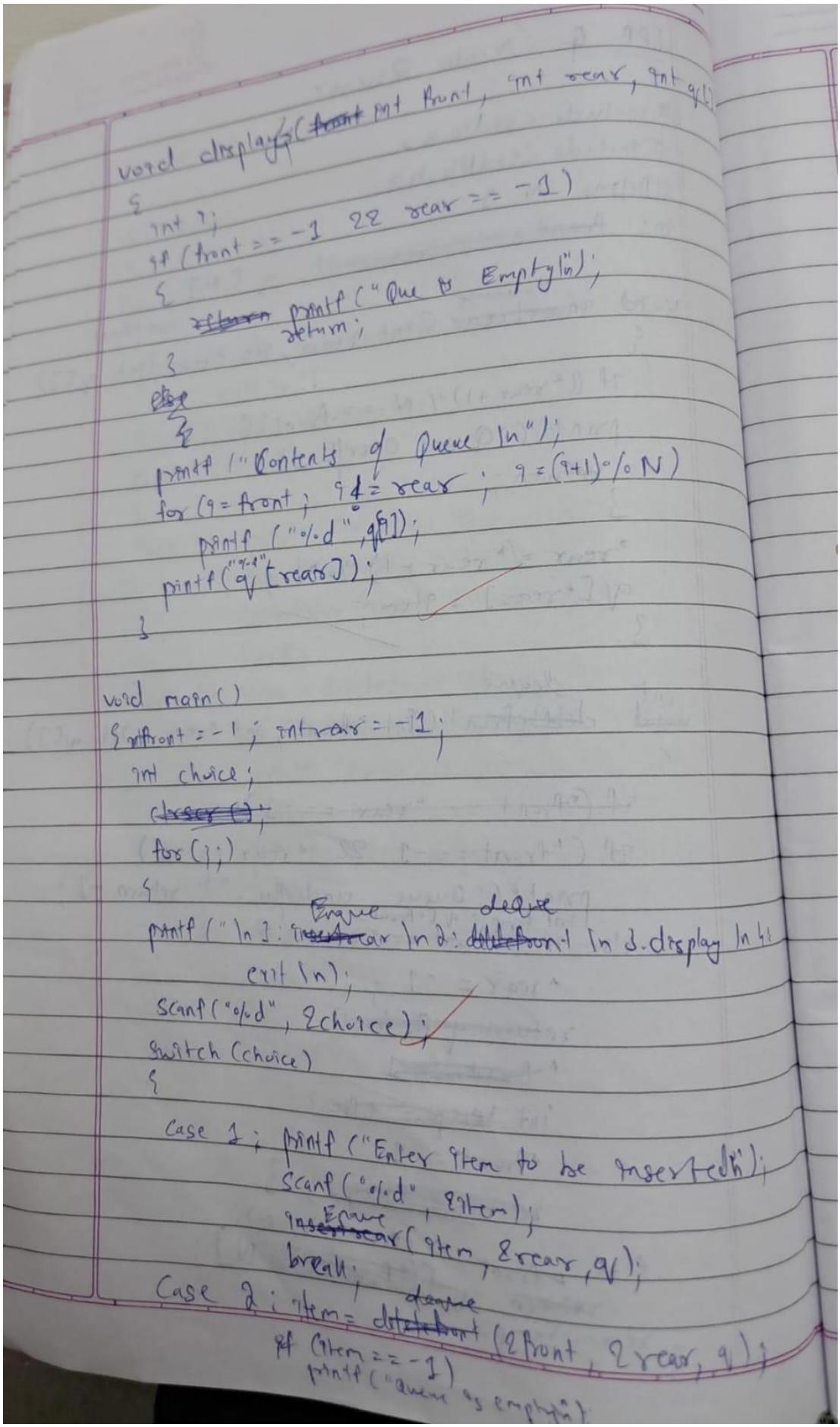
Output 3

Program 4

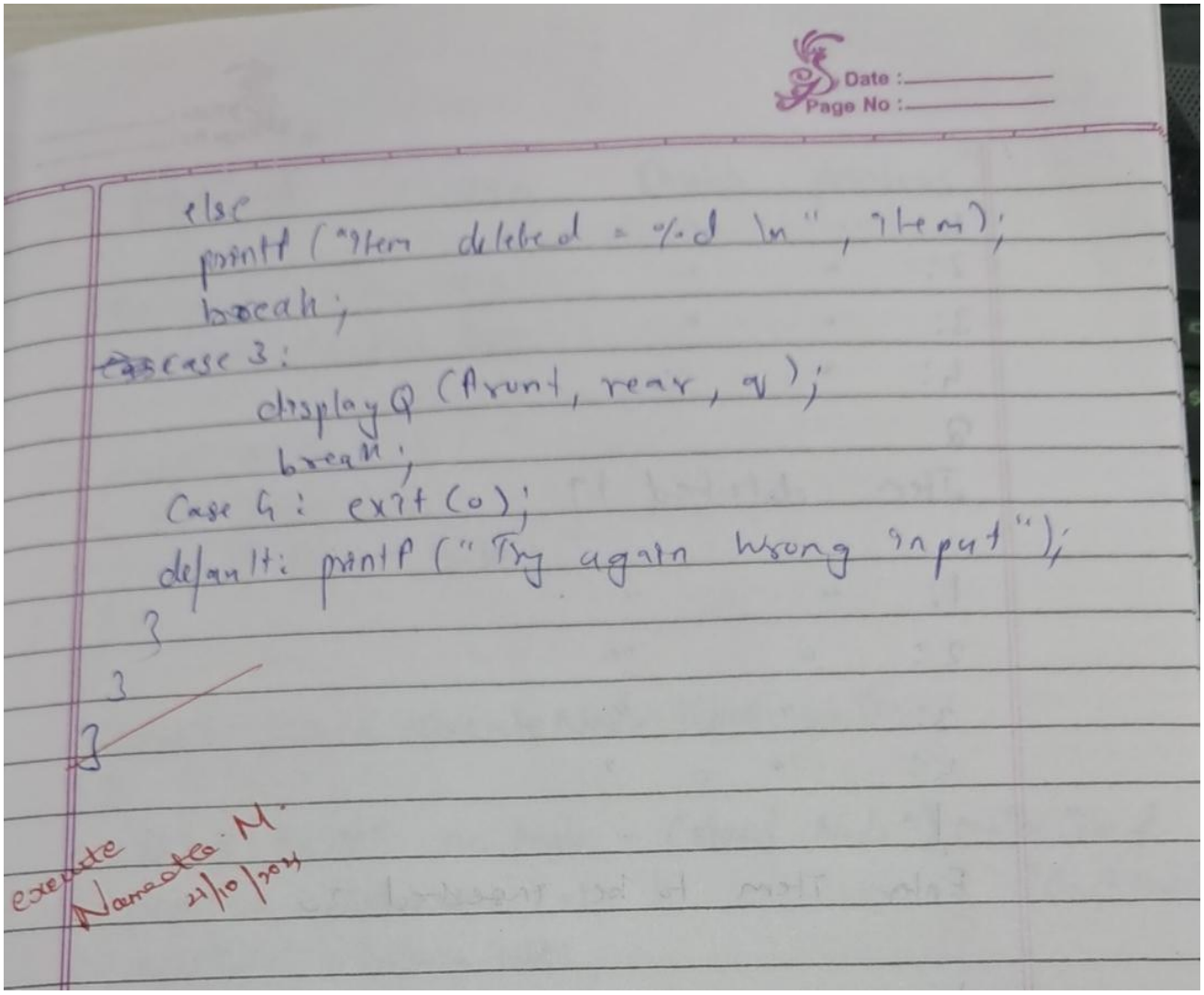
WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & Display.
The program should print appropriate messages for queue empty and queue overflow conditions.



```
LAB 5 Circular Queue

#include <stdio.h>
#include <stdlib.h>
#Define N 5
int front = -1, rear = -1, q[N]

void enqueue(int item, int *rear, int *front, int q[])
{
  if ((*rear+1)%N == front){
  printf("Queue Overflow")
  return;
  }
  *rear = (*rear + 1) % N;
  q[*rear] = item;
}

int dequeue(int *front, int *rear, int q[])
{
  if (*front == -1 && *rear == -1)
    printf("Queue underflow"); return -1;
  int temp = q[*front];
  else if (*front == *rear) {
    *rear = -1;
    return q[(*front)
    *front = -1
    int temp = *front;
    *front = -1;
    return q[temp]; }
  else
    *front = ((*front+1)%N);
    return q[(*front+1)%N];
  return temp;
}
```



```
void display(int front, int rear, int q[])
{
 int i;
 if (front == -1 && rear == -1)
 {
  printf("Que is Empty\n");
  return;
 }
 printf("Contents of Queue \n");
 for (i = front; i != rear ; i = (i+1) % N)
  printf("%d", q[i]);
 printf("%d", q[rear]);
}

void main()
{ int front = -1; int rear = -1;
 int choice;
 for(;;)
 {
 printf("\n 1: Enque \n 2: deQue \n 3. display \n 4:
        exit \n");
 scanf("%d", &choice);
 switch (choice)
 {
  case 1: printf("Enter item to be inserted");
          scanf("%d", &item);
          Enque(item, &rear, q);
          break;
  Case 2: item = dequeue(&front, &rear, q);
          if (item == -1)
          printf("queue is empty");
```

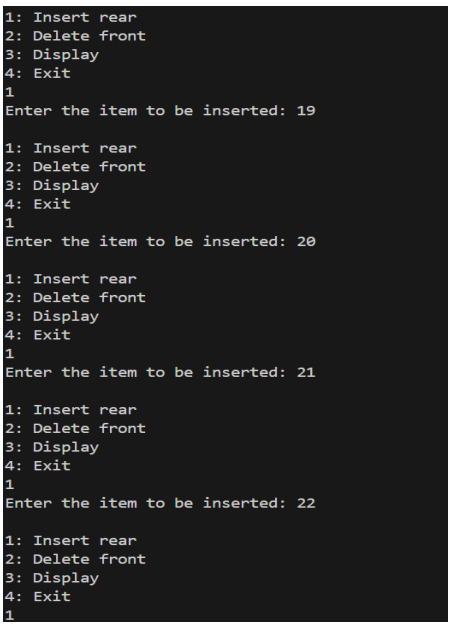


Code:

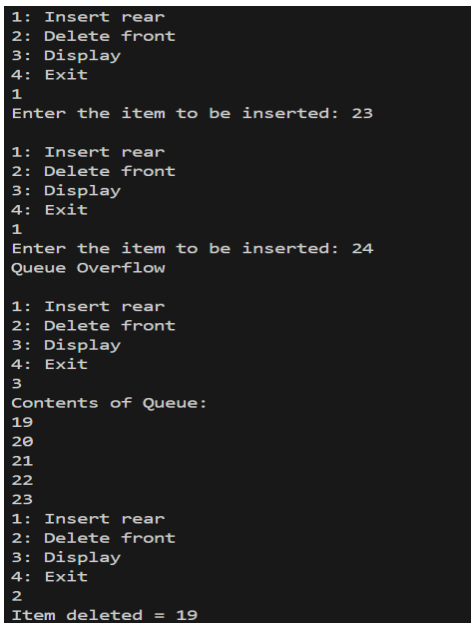
```
#include <stdio.h>
#include <stdlib.h>

#define N 5

int q[N];

void Enqueue(int item, int *rear,int *front , int q[]) {
   if (((*rear+1)%N) == *front) {
      printf("Queue Overflow \n");
      return;
   }
   if (*front == -1)
   {
      *front = 0;
```

```
    }

    *rear = ((*rear + 1)%N);
    q[*rear] = item;
}

int Deque(int *front, int *rear, int q[]) {
    if (*front == -1 && *rear == -1) {
        return -1;
    }
    int temp = q[*front];
    if(*front == *rear){
        *rear = -1;
        *front = -1;
    }
    else{
        *front = ((*front+1)%N);
    }
    return temp;
}

void displayQ(int front, int rear, int q[]) {
    if (front == -1 &&  rear == -1) {
        printf("Queue is empty \n");
        return;
    }
    printf("Contents of Queue:\n");
    for (int i = front; i != rear; i=(i+1)%N) {
        printf("%d \n", q[i]);
    }
    printf("%d",q[rear]);
}

int main() {
    int choice;
    int rear = -1;
    int front = -1;
    int item;
    for (;;) {
        printf("\n1: Enque\n2: Deque1\n3: Display\n4: Exit\n");
        scanf("%d", &choice);
        switch (choice) {
```

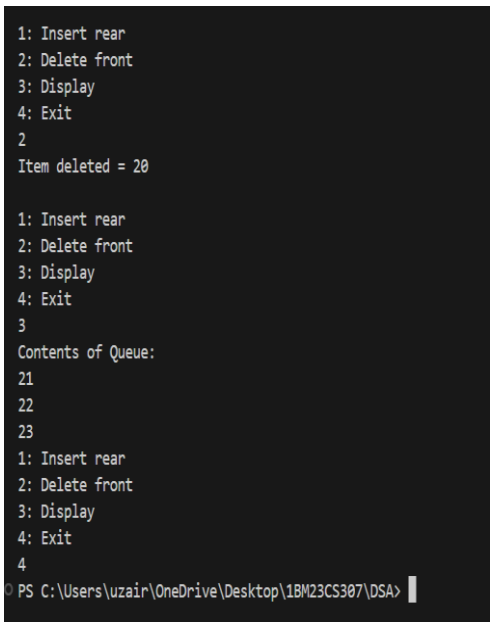
```
            case 1:
                printf("Enter the item to be inserted: ");
                scanf("%d", &item);
                Enqueue(item, &rear, &front , q);
                break;
            case 2:
                item = Deque(&front, &rear, q);
                if (item == -1)
                    printf("Queue is empty\n");
                else
                    printf("Item deleted = %d \n", item);
                break;
            case 3:
                displayQ(front, rear, q);
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice, please try again.\n");
        }
    }
    return 0;
}
```



Output 1



Output 2



Output 3

Program 5

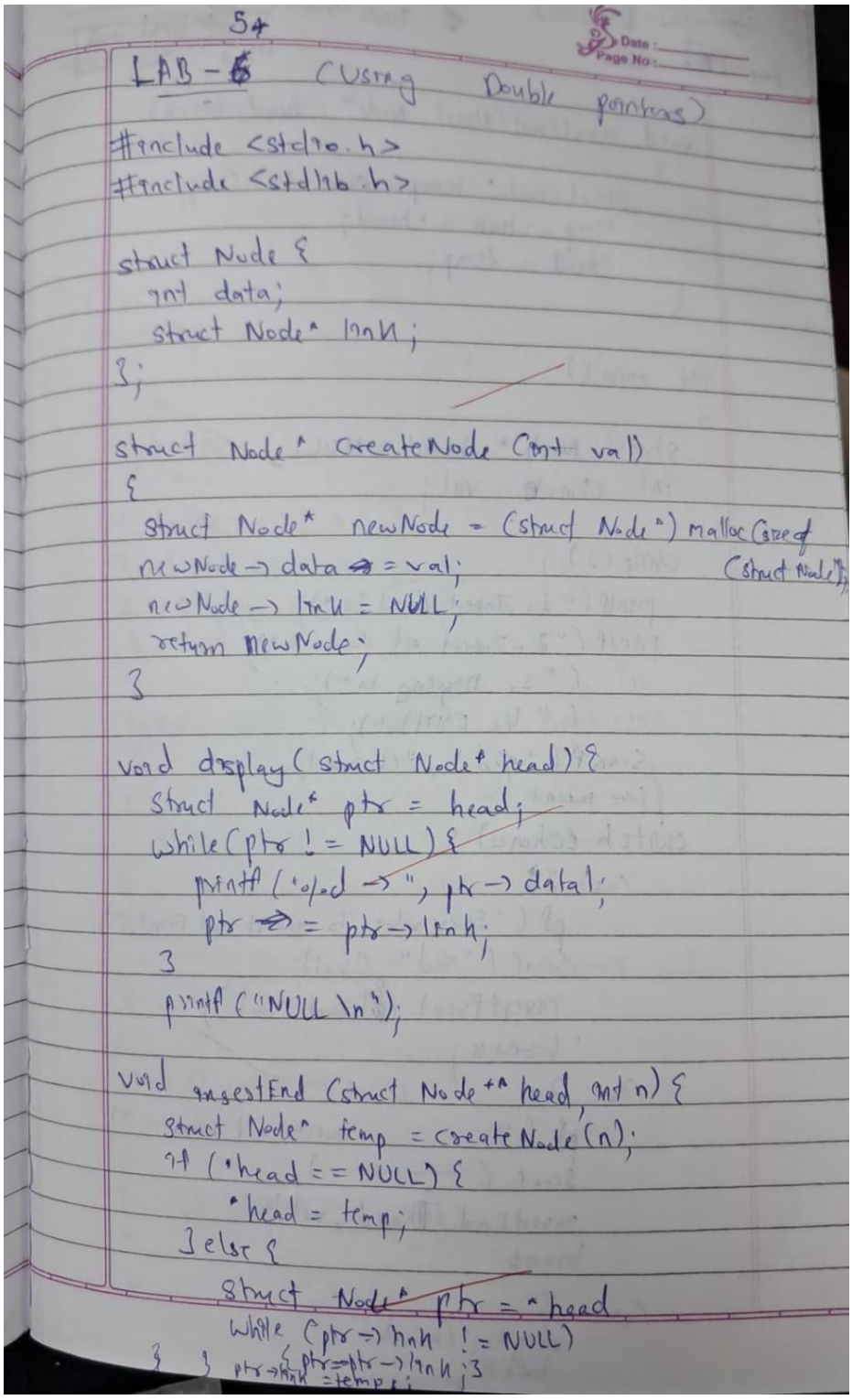
Write a program to Implement Singly Linked List with following operations

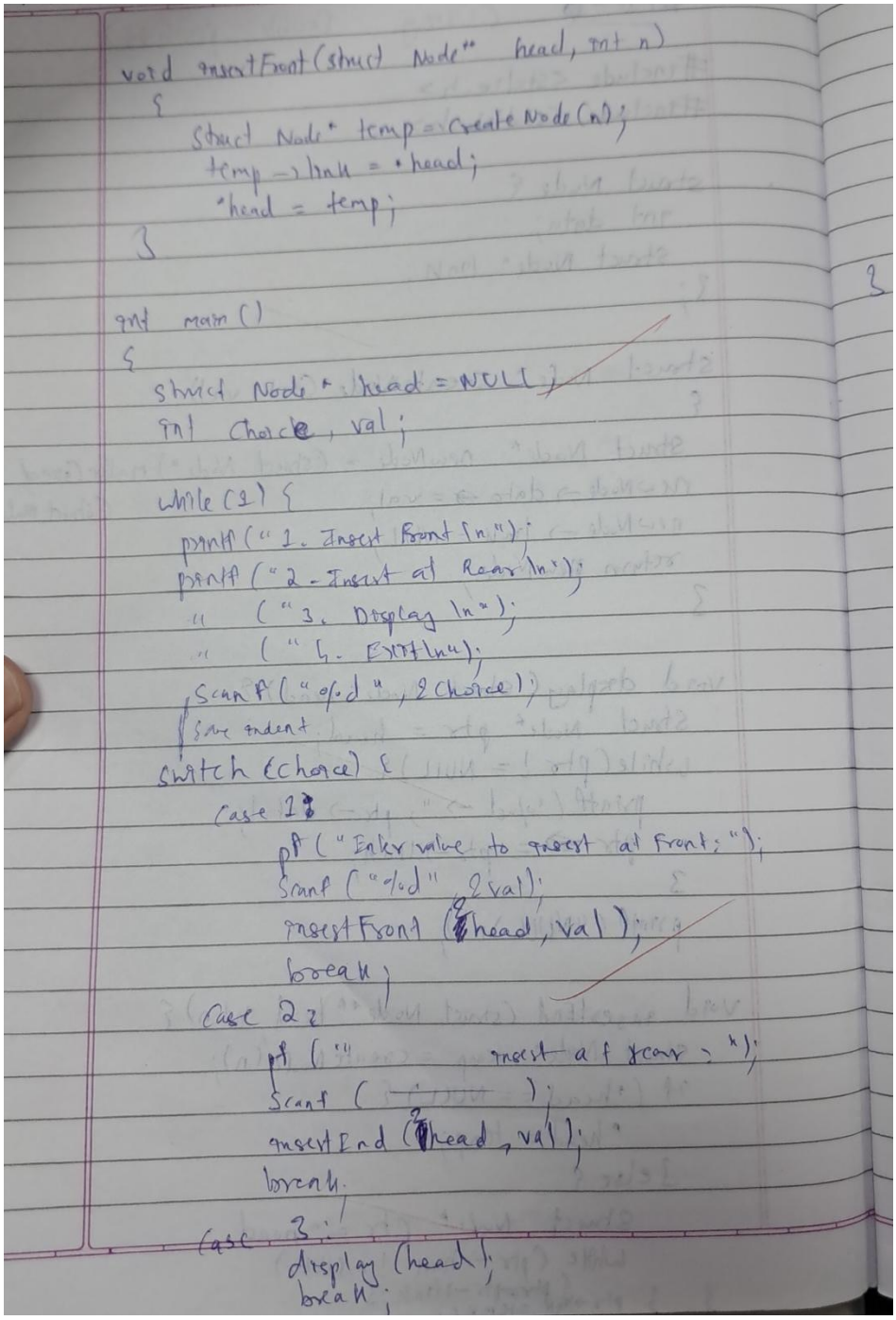
a) Create a linked list.

b) Insertion of a node at **first position** and **at end of list.**

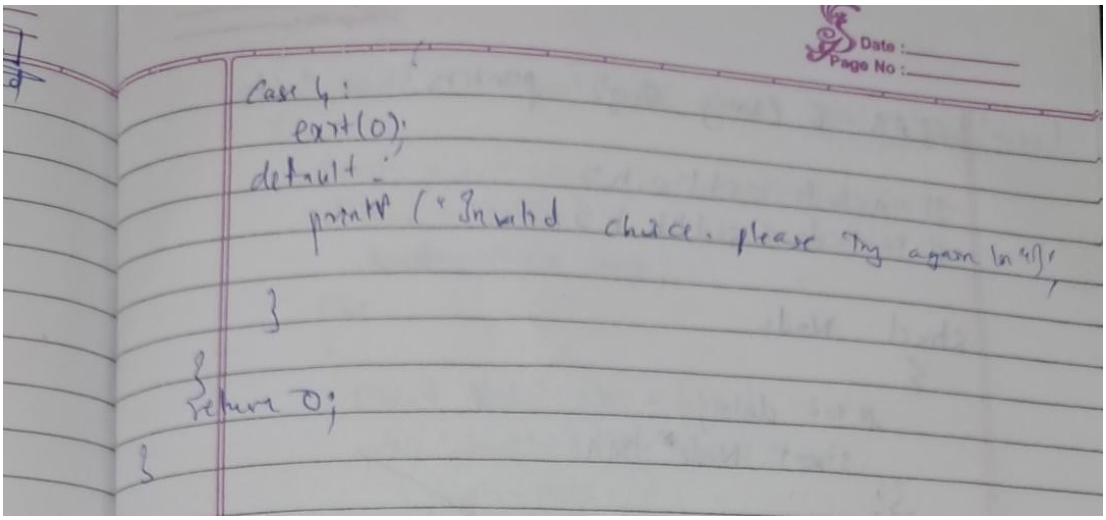
Display the contents of the linked list.

Leetcode problem no.20 (Valid parantheses)





```
void insertFront(struct Node** head, int n)
{
    struct Node* temp = createNode(n);
    temp->link = *head;
    *head = temp;
}

int main()
{
    struct Node* head = NULL;
    int choice, val;
    while(1) {
    printf("1. Insert Front \n");
    printf("2 - Insert at Rear \n");
    "   ("3. Display \n");
    "   ("4. Exit\n");
    scanf("%d", &choice);
    save indent
switch(choice) {
    case 1:
        pf("Enter value to insert at Front: ");
        scanf("%d", &val);
        insertFront(&head, val);
        break;
    case 2:
        pf("          insert at rear : ");
        scanf(          );
        insertEnd(&head, val);
        break;
    case 3:
        display(head);
        break;
```

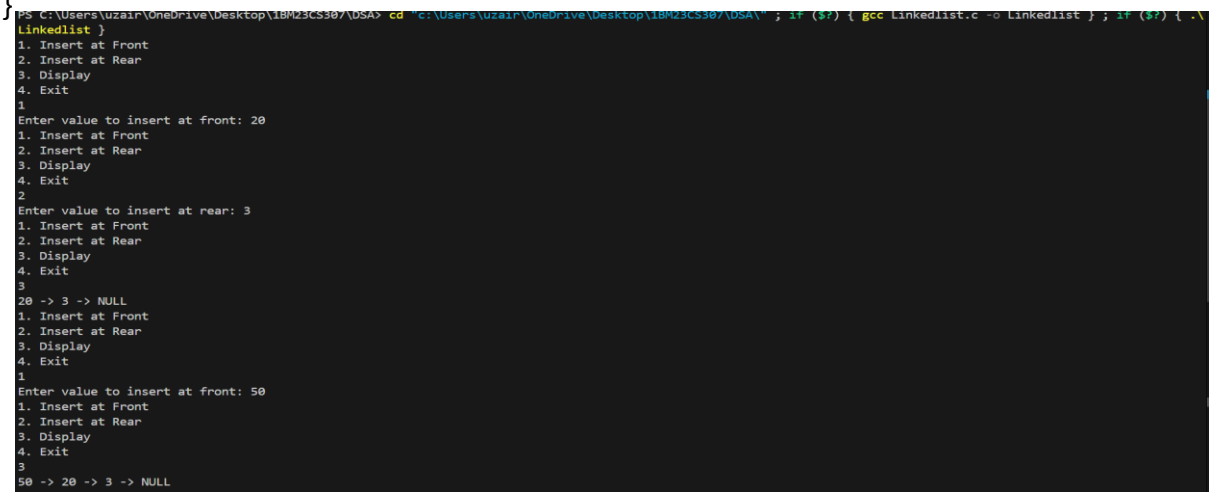


```
    case 4:
        exit(0);
    default:
        printf("Invalid choice. please Try again \n");
    }
}
return 0;
}
```

Code:

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

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

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

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

void insertEnd(struct Node** head, int n) {
    struct Node* temp = createNode(n);
    if (*head == NULL) {
        *head = temp;  // If the list is empty, the new node becomes the head
    } else {
        struct Node* ptr = *head;
        while (ptr->link != NULL) {
            ptr = ptr->link;
        }
        ptr->link = temp;  // Add the new node at the end
    }
}

void insertFront(struct Node** head, int n) {
    struct Node* temp = createNode(n);
    temp->link = *head;
    *head = temp;  // The new node becomes the head of the list
}
```

```c
int main() {
    struct Node* head = NULL;
    int choice, val;

    while (1) {
        printf("1. Insert at Front\n");
        printf("2. Insert at Rear\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert at front: ");
                scanf("%d", &val);
                insertFront(&head, val);
                break;
            case 2:
                printf("Enter value to insert at rear: ");
                scanf("%d", &val);
                insertEnd(&head, val);
                break;
            case 3:
                display(head);
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
}
```



```c
#include <string.h>

bool isValid(char* s) {
    int n = strlen(s);
    char stack[n];
    int top = -1;
    int i = 0;
    while (i < n)
    {
        char c = s[i];
        if (c == '(' || c == '{' || c == '[')
            stack[++top] = c;
        else
        {
            if (top == -1)
                return false;
            char topChar = stack[top--];
            if ((c == ')' && topChar != '(') ||
                (c == '}' && topChar != '{') ||
                (c == ']' && topChar != '['))
                return false;
        }
        i++;
    }
    return top == -1;
}
```

Leet Code Valid Parenthesis

Program 6

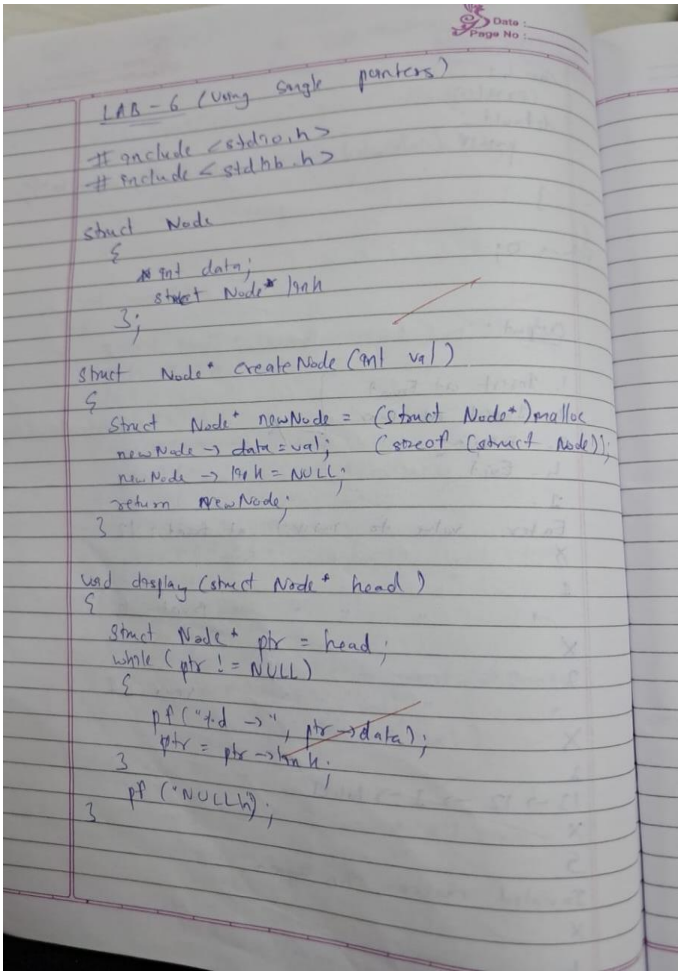
Write a Program to Implement Singly Linked List with following operations

a) Create a linked list.

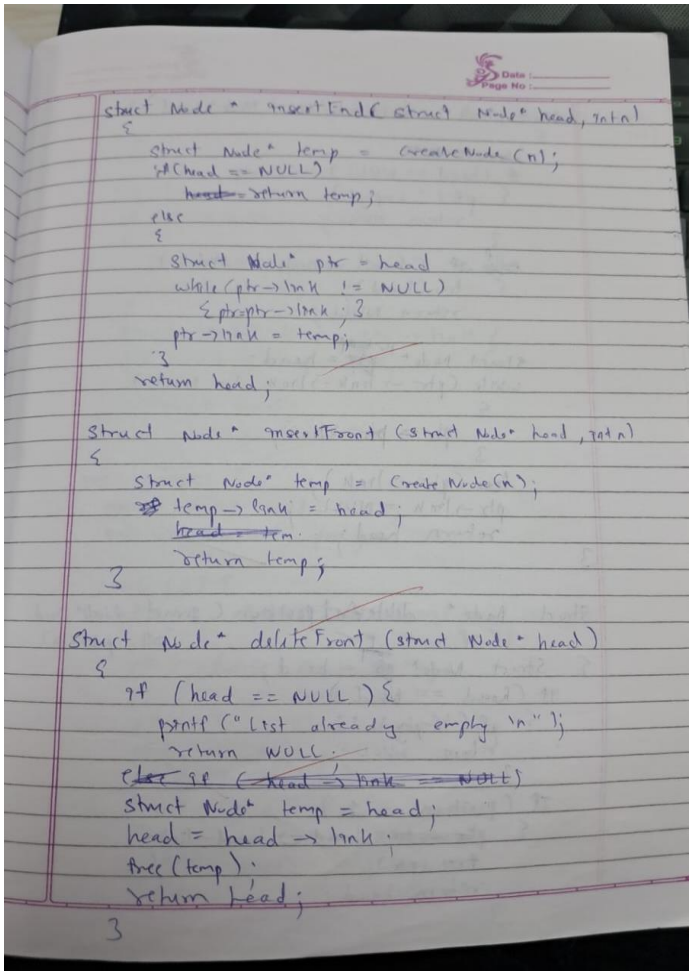
b) Deletion of first element, specified element and last element in the list.

c) Display the contents of the linked list.

Leetcode Problem -- 739 (Daily Temperature)



```
LAB - 6 (Using Single pointers)

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

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

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

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



```
struct Node * insertEnd( struct Node* head, int n)
{
    struct Node* temp = createNode (n);
    if(head == NULL)
        return temp;
    else
    {
        struct Node* ptr = head
        while(ptr -> link != NULL)
            {ptr=ptr -> link ;}
        ptr -> link = temp;
    }
    return head;
}

struct Node * insertFront (struct Node* head, int n)
{
    struct Node* temp = createNode(n);
    temp -> link = head;
        return temp;
}

struct Node* deleteFront (struct Node* head)
{
    if (head == NULL ){
        printf ("List already empty \n");
        return NULL;
    else if (head -> link == NULL)
    struct Node* temp = head;
    head = head -> link;
    free (temp);
    return head;
}
```

```
struct Node * deleteEnd (struct Node * head)
{
    if (head == NULL)
    { printf ("Empty");
      return NULL;
    }
    if (head -> link == NULL)
    { free (head);
      return NULL;
    }
    struct Node * ptr = head;
    while (ptr -> link -> link != NULL)
    {
        ptr = ptr -> link;
    }
    free (ptr -> link);
    ptr -> link = NULL;
    return head;
}

struct Node * deleteAtposition (struct Node * head,
                                int position)
{ struct Node * ptr = head;
  if (head == NULL)
  { printf ("Empty \n");
    return NULL;
  }
  if (position == 1)
  { ptr = head -> link;
    free (ptr);
    return head;
  }
}
```

```
for (int i=1; i< position-1 && ptr->link != NULL; i++)
{
    ptr = ptr->link;
}
if (ptr->link == NULL)
{
    pf("Out of bounds.\n");
    return head;
}
struct Node* temp = ptr->link;
ptr->link = temp->link;
free(temp);
return head;
}

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

    while (1) {
        pf("1. Insert at Front\n");
        pf("2.      "  "  Rear\n");
        pf("3. Delete Front\n");
        pf("4.      "  Rear\n");
        pf("5.      "  at Position\n");
        pf("6. Display\n");
        pf("7. Exit\n");
        scanf("%d", &choice);

        switch(choice)
        {
```

```
case 1:
    printf("Enter value to insert at front:");
    scanf("%d", &val);
    head = insertFront(head, val);
    break;

case 2:
    pf("                    insert at rear:");
    scanf(_____);
    head = insertEnd(head, val);
    break;

case 3:
    head = deleteFront(head);
    break;

case 4:
    head = deleteEnd(head);
    break;

case 5:
    printf("Enter position to delete:");
    scanf("%d", &pos);
    head = deleteAtPosition(head, pos);
    break;
case 6:
    display(head);
    break;
case 7:
    exit(0);
default:
    printf("\n Invalid choice. Try Again.\n")
}
return 0;
```

Code:

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

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

// Function to create a new node
struct Node* createNode(int val) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = val;
    newNode->link = NULL;
    return newNode;
}

// Function to display the linked list
void display(struct Node* head) {
    struct Node* ptr = head;
    while (ptr != NULL) {
        printf("%d -> ", ptr->data);
        ptr = ptr->link;
    }
    printf("NULL\n");
}

// Function to insert at the end of the list
struct Node* insertEnd(struct Node* head, int n) {
    struct Node* temp = createNode(n);

    if (head == NULL) {
        return temp;  // If the list is empty, the new node becomes the head
    } else {
        struct Node* ptr = head;
        while (ptr->link != NULL) {
            ptr = ptr->link;
        }
        ptr->link = temp;  // Add the new node at the end
    }
    return head;
}

// Function to insert at the front of the list
struct Node* insertFront(struct Node* head, int n) {
    struct Node* temp = createNode(n);
    temp->link = head;  // The new node points to the current head
    return temp;        // Return the new node as the new head of the list
```

```
}

// Function to delete the front node
struct Node* deleteFront(struct Node* head) {
    if (head == NULL) {
        printf("The list is already empty.\n");
        return NULL;
    }
    struct Node* temp = head;
    head = head->link;   // Move head to the next node
    free(temp);          // Free the old head
    return head;
}

// Function to delete the end node
struct Node* deleteEnd(struct Node* head) {
    if (head == NULL) {
        printf("The list is already empty.\n");
        return NULL;
    }
    if (head->link == NULL) { // Only one node in the list
        free(head);
        return NULL;
    }
    struct Node* ptr = head;
    while (ptr->link->link != NULL) {
        ptr = ptr->link;
    }
    free(ptr->link);      // Free the last node
    ptr->link = NULL;     // Set the second last node's link to NULL
    return head;
}

// Function to delete a node at a specified position
struct Node* deleteAtPosition(struct Node* head, int position) {
    struct Node* ptr = head;
    if (head == NULL) {
        printf("The list is empty.\n");
        return NULL;
    }
    if (position == 1) { // Delete the first node
        head=head->link;
        free(ptr);
        return head;
    }
    for (int i = 1; i < position - 1 && ptr->link != NULL; i++) {
        ptr = ptr->link;
    }
    if (ptr->link == NULL) {
```

```
        printf("Position out of bounds.\n");
        return head;
    }
    struct Node* temp = ptr->link;
    ptr->link = temp->link;
    free(temp);          // Free the node at the specified position
    return head;
}

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

    while (1) {
        printf("1. Insert at Front\n");
        printf("2. Insert at Rear\n");
        printf("3. Delete Front\n");
        printf("4. Delete Rear\n");
        printf("5. Delete at Position\n");
        printf("6. Display\n");
        printf("7. Exit\n");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert at front: ");
                scanf("%d", &val);
                head = insertFront(head, val);
                break;
            case 2:
                printf("Enter value to insert at rear: ");
                scanf("%d", &val);
                head = insertEnd(head, val);
                break;
            case 3:
                head = deleteFront(head);
                break;
            case 4:
                head = deleteEnd(head);
                break;
            case 5:
                printf("Enter position to delete: ");
                scanf("%d", &pos);
                head = deleteAtPosition(head, pos);
                break;
            case 6:
                display(head);
                break;
            case 7:
```

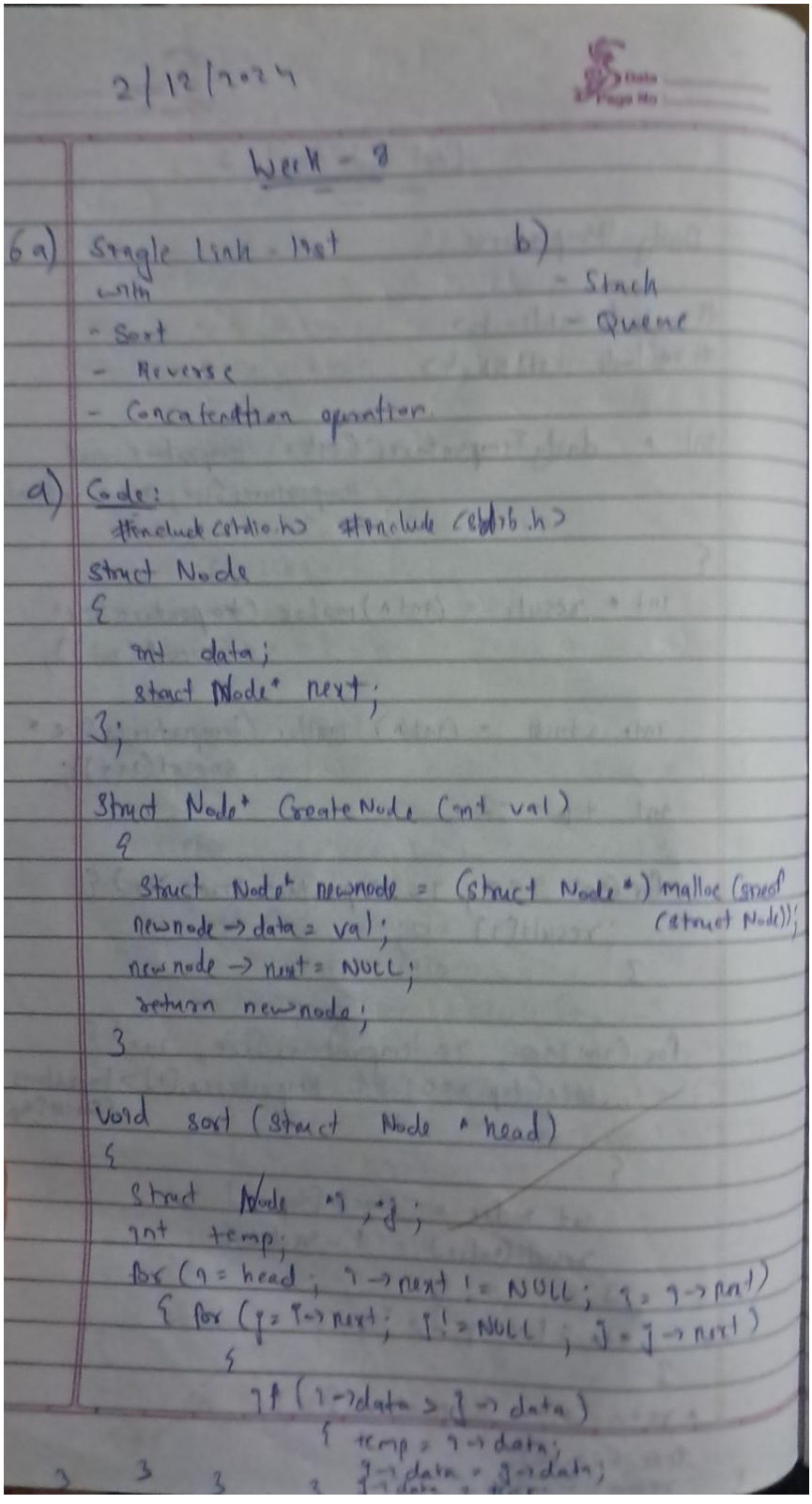
```
            exit(0);
        default:
            printf("Invalid choice. Please try again.\n");
    }
  }
  return 0;
}
```

```c
/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
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 !=-1 && temperatures[i] > temperatures[stack[top]]) {
            int j = stack[top--];
            answer[j] = i - j;
        }
        stack[++top] = i;

    free(stack);
    return answer;

}
```

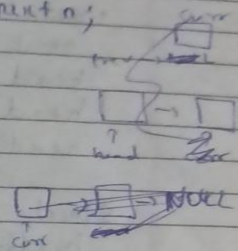
Leet Code Daily Temperatures

Program 7

a) Write a Program to Implement Single Link List with following operations:
(i)Sort the linked list,
(ii) Reverse the linked list,
(iii)Concatenation of two linked lists.



```
Struct Node *
reverse (struct Node * head)
{
    Struct Node *prev, *curr, *nextn;
    prev = NULL;
    curr = head;
    nextn = head;

    while (nextn != NULL)
    {
        nextn = nextn -> next;
        curr -> next = prev;
        prev = curr;
        curr = nextn;
    }
    return prev;
}

Struct Node*
Concat (struct Node *head1, struct Node *head2)
{   if (head1 == NULL) return head2;
    Struct Node *temp2;
    temp1 = head1;
    while (temp1 -> next != NULL)
    {
        temp1 = temp1 -> next
    }
    temp1 -> next = head2;
```



```
Void display (struct Node * head)
{
    struct Node * ptr = head;
    while (ptr != NULL)
    { pf ("%d -> " ptr -> data);
      ptr = ptr -> next; }
    pf ("NULL \n");
}
Void main ()
{
    struct Node * head = NULL;
    head -> data = 4;
    head -> next = CreateNode (2);
    head -> next -> next = CreateNode (9);
    head -> next -> next -> next = CreateNode (1);

    Struct Node * head2 = NULL;
    head2 -> data = 5;
    head2 -> next = CreateNode (3);
    head2 -> next -> next = CreateNode (8);

    display (head);
    display (head2);

    sort (head);
    display (head);

    head2 = reverse (head2);
    display (head2);

    head = Concat (head, head2);
    display (head);
}
```

Code:

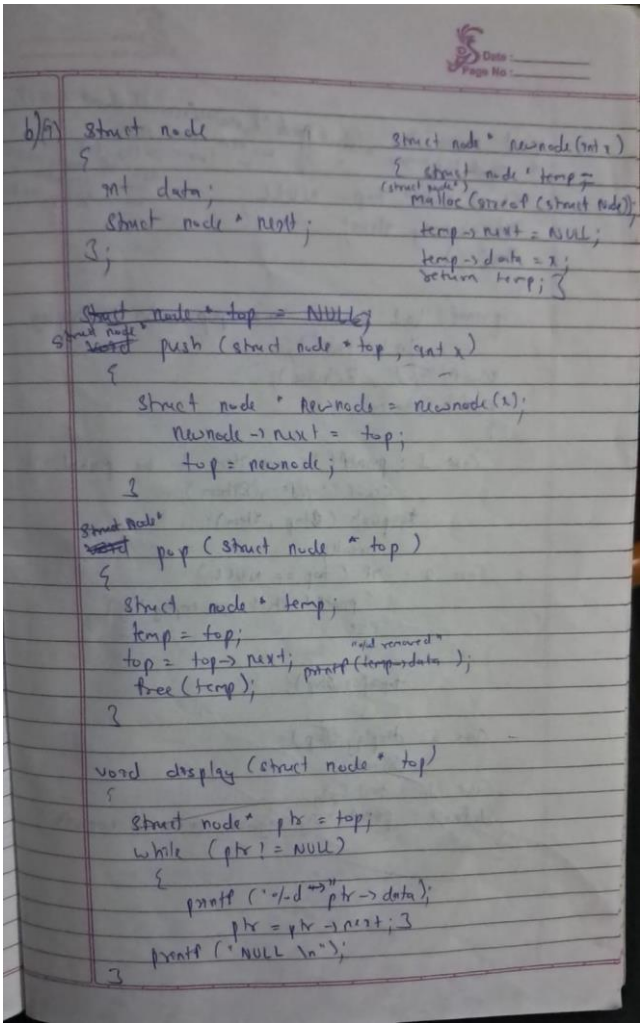
```
#include <stdio.h>
#include <stdlib.h>

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

struct Node* CreateNode(int val) {
    struct Node* newnode = (struct Node*)malloc(sizeof(struct Node));
    newnode->data = val;
    newnode->next = NULL;
    return newnode;
}

void sort(struct Node* head) {
    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;
            }
        }
    }
}

struct Node* reverse(struct Node* head) {
    struct Node *prev = NULL, *curr = head, *nextn = NULL;
    while (curr != NULL) {
        nextn = curr->next;
        curr->next = prev;
        prev = curr;
        curr = nextn;
    }
    return prev;
}

struct Node* Concat(struct Node* head1, struct Node* head2) {
    if (head1 == NULL) return head2;

    struct Node* temp1 = head1;
    // Traverse to the last node of the first list
```

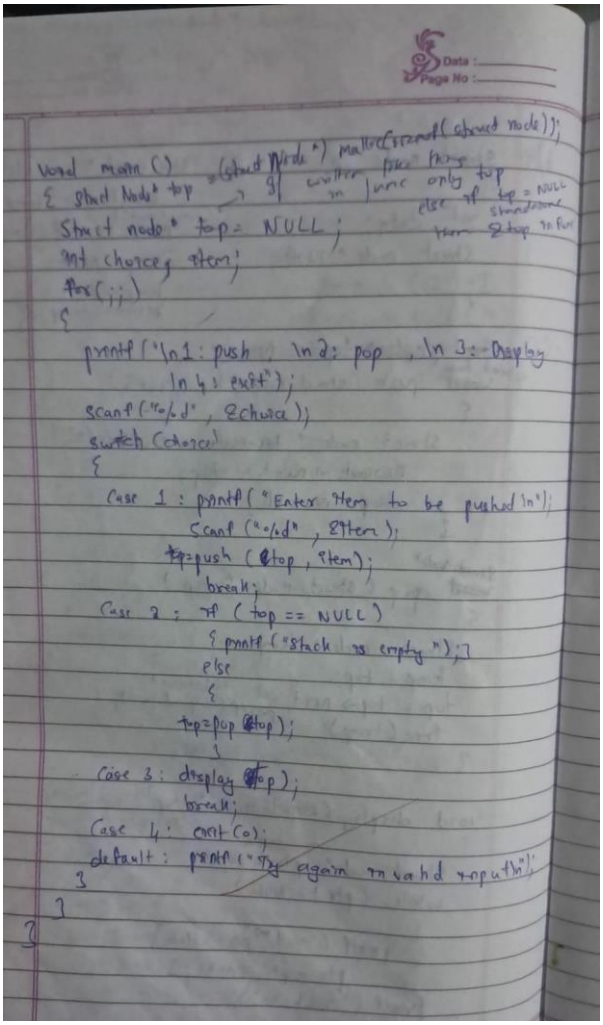
```
    while (temp1->next != NULL) {
        temp1 = temp1->next;
    }
    temp1->next = head2;
    return head1;
}

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

int main() {
    struct Node* head = CreateNode(4);
    head->next = CreateNode(2);
    head->next->next = CreateNode(9);
    head->next->next->next = CreateNode(1);

    struct Node* head2 = CreateNode(5);
    head2->next = CreateNode(3);
    head2->next->next = CreateNode(8);
    display(head);
    display(head2);

    sort(head);
    display(head);

    head2 = reverse(head2);
    display(head2);

    head = Concat(head, head2);
    display(head);

    return 0;
}
```

```
PS C:\Users\uzair\OneDrive\Desktop\1BM23CS307> cd "c:\Users\uzair\OneDrive\Desktop\1BM23CS307\DSA\Lab 8\" ;
if ($?) { gcc SLLOperations.c -o SLLOperations } ; if ($?) { .\SLLOperations }
4 ->2 ->9 ->1 ->NULL
5 ->3 ->8 ->NULL
1 ->2 ->4 ->9 ->NULL
8 ->3 ->5 ->NULL
1 ->2 ->4 ->9 ->8 ->3 ->5 ->NULL
PS C:\Users\uzair\OneDrive\Desktop\1BM23CS307\DSA\Lab 8>
```

Output

b)Write a Program to Implement Single Link List to simulate
(i)Stack
(ii)Queue Operations.



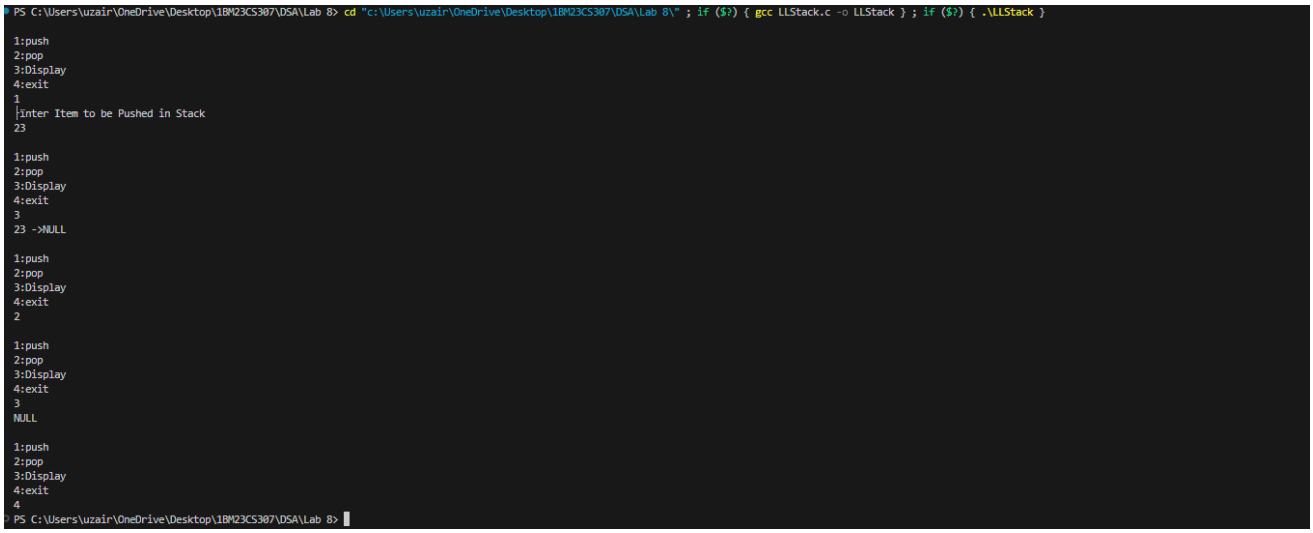


(i) Code:

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

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

struct Node* CreateNode(int val) {
```

```
    struct Node* newnode = (struct Node*)malloc(sizeof(struct Node));
    newnode->data = val;
    newnode->next = NULL;
    return newnode;
}


struct Node* push(struct Node*top,int x)
{
    struct Node* newnode = CreateNode(x);
    newnode->next = top;
    top = newnode;
    return newnode;
}

struct Node* pop(struct Node* top)
{
    struct Node* temp;
    temp = top;
    top = top->next;
    free(temp);
    return top;

}

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

void main()
{
    struct Node* top = (struct Node*)malloc(sizeof(struct Node));
    top = NULL;
    int choice,item;
    for(;;)
    {
        printf("\n1:push \n2:pop \n3:Display \n4:exit\n");
        scanf("%d",&choice);
        switch (choice)
        {
        case 1:
            printf("Ënter Item to be Pushed in Stack \n");
            scanf("%d",&item);
            top = push(top,item);
```

```
            break;
        case 2:
            if (top == NULL)
            {
                printf("Stack is Empty");
            }
            else
            {
                top =pop(top);
            }
            break;
        case 3:
            display(top);
            break;
        case 4:
            exit(0);
        default:
        printf("Try again invalid input \n");
        break;
        }
    }
}
```



Output (i)

```
(ii) struct node
{ same as
  stack
}

struct node* newnode (int x)
{
  Same as stack
}
struct node*
void dequeue ( struct node * front , struct node* rear)
{
  struct node * temp = front;
  temp = front;
  if (front == NULL  && rear == NULL)
      printf ("Underflow \n"); return NULL;
  else if (front == rear)
  { temp = front;
    front = NULL;
    rear = NULL;
    printf ("%d removed", temp -> data);
    free (temp);
  }
  else
  {
    front = front -> next;
    free (temp);
    return front;
  }
}
```

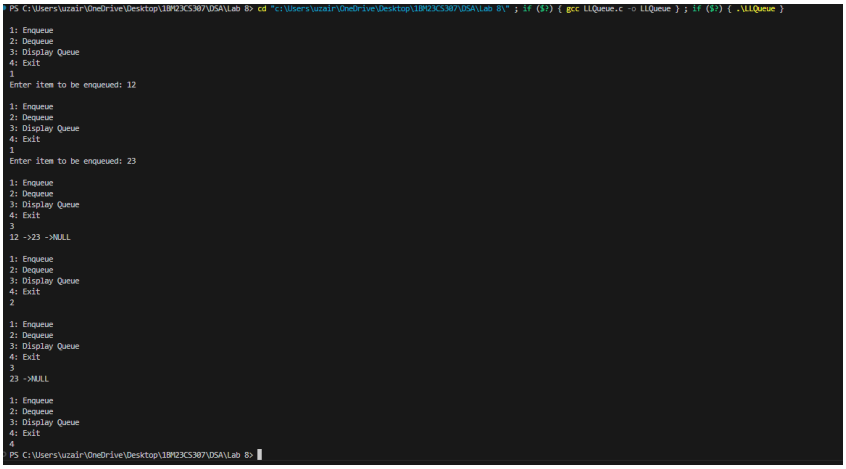
```
struct node*
void enque ( struct node* front , struct node* rear,
                int x)
{
  struct node* temp = newnode(x);
  if (front == NULL && rear == NULL)
  {
    front = temp; return temp;
    rear = temp;
    return;
  }
  else
  {
    rear -> next = temp;
    rear = temp;
    rear = temp; rear
    return newnode temp;
  }
}

                    struct node* front
void display()
{
  same as stack;
}

void front ()
                    struct node * front
{
  pf ("%d", front -> data);
}

                    struct node * rear
void rear()
{ printf ("%d", rear -> data);
```

```
Void main ()
{
  Struct node* front = NULL;
  Struct node * rear = NULL;
  int choice, data;
  for (;;)
  {
    printf ("\n 1: enque , \n 2: deque \n 3: Display
          \n 4: front \n 5: Rear \n 6: exit");
    Scanf ("%d", &choice);
    switch (choice)
    {
      Case 1 : printf ("Enter item to enque ");
               Scanf ("%d", &data);
               rear=enque ( front, rear, data);
               break;
      Case 2 : front = deque (front, rear)
               if (front == NULL) { front = rear = NULL; }
               break;
      Case 3: display (front);
               break;
      Case 4: front(front);
               break;
      Case 5 : rear(rear);
               break;
      Case 6 : exit(0);
      default : printf (" Try again invalid input");
    }
  }
}
```

(ii) Code:

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

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

struct Node* CreateNode(int val) {
    struct Node* newnode = (struct Node*)malloc(sizeof(struct Node));
    newnode->data = val;
    newnode->next = NULL;
    return newnode;
}

struct Node* enqueue(struct Node* rear, int x) {
    struct Node* newnode = CreateNode(x);
    if (rear == NULL) {
        return newnode;
    }
    rear->next = newnode;
    return newnode;
}

struct Node* dequeue(struct Node* front) {
    if (front == NULL) {
        printf("Queue is empty\n");
        return NULL;
    }
    struct Node* temp = front;
    front = front->next;
    free(temp);
    return front;
}

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

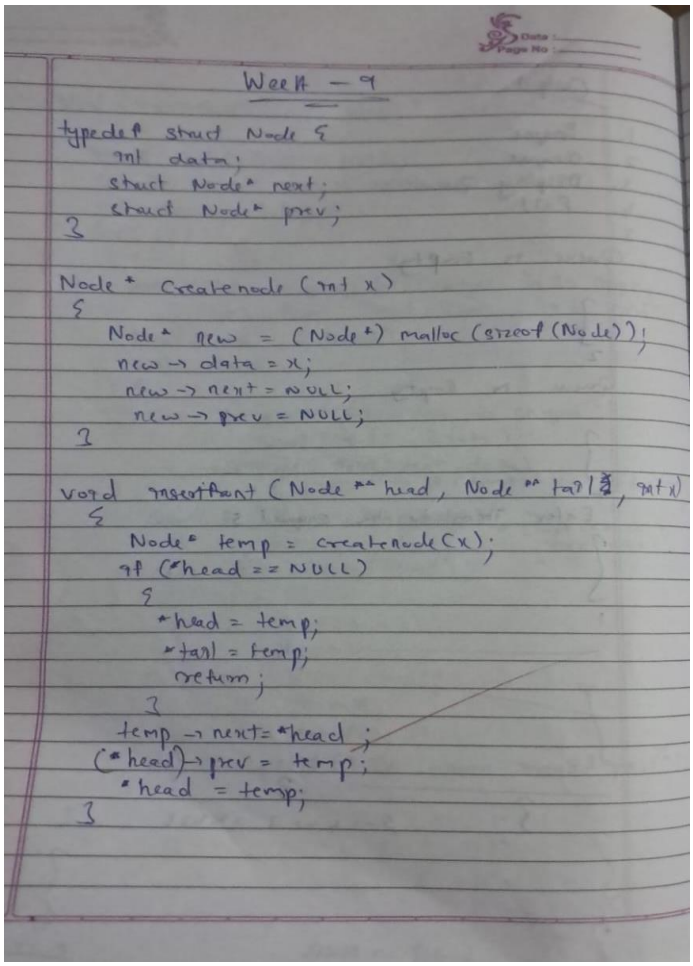
```
int main() {
    struct Node* front = NULL;
    struct Node* rear = NULL;
    int choice, item;

    for (;;) {
        printf("\n1: Enqueue\n2: Dequeue\n3: Display Queue\n4: Exit\n");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
            printf("Enter item to be enqueued: ");
            scanf("%d", &item);
            rear = enqueue(rear, item);  // Enqueue operation, rear updated
            if (front == NULL) {
                front = rear;
            }
            break;
        case 2:
            front = dequeue(front);  // Dequeue operation, front updated
            if (front == NULL) {
                rear = NULL;
            }
            break;
        case 3:
            display(front);
            break;
        case 4:
            exit(0);  // Exit the program
        default:
            printf("Invalid input. Please try again.\n");
            break;
        }
    }
}
```



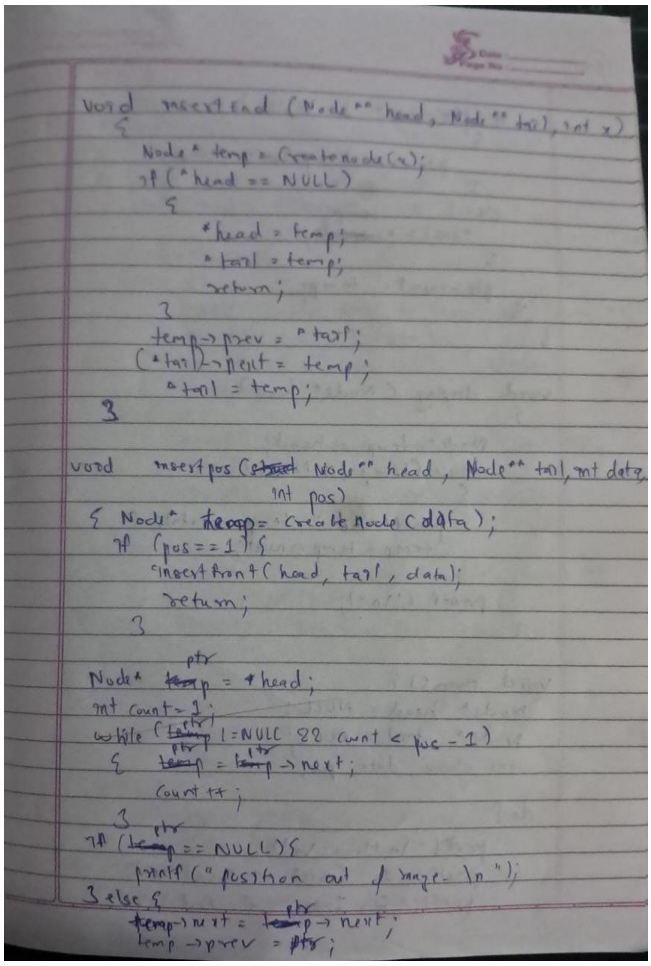
Output (ii)

Program 8

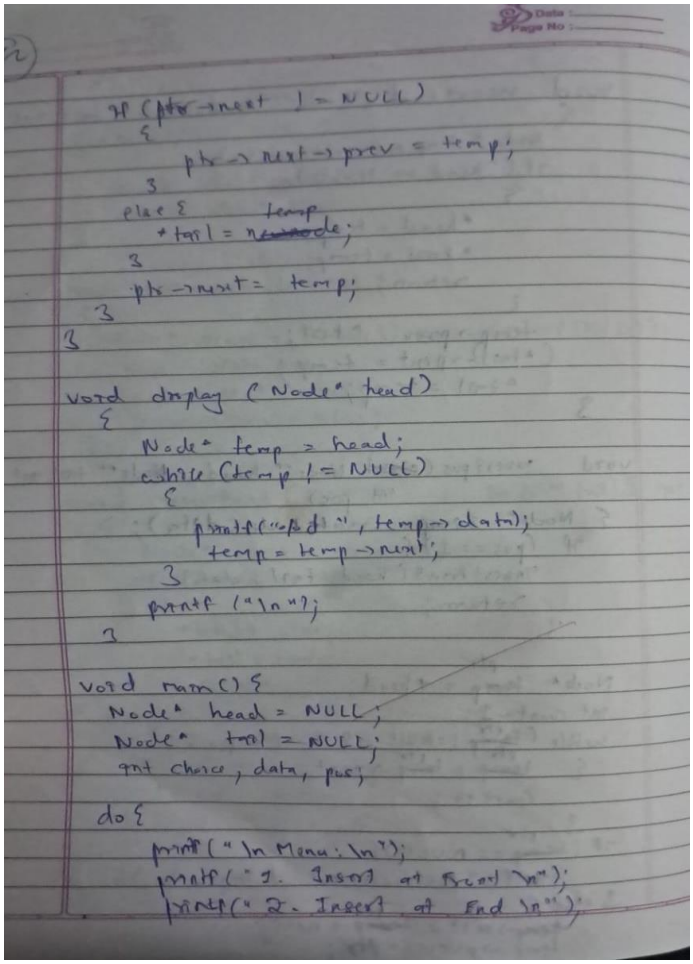
Write a Program to Implement doubly link list with primitive operations
a) Create a doubly linked list.
b) Insert a new node at the beginning.
c) Insert the node based on a specific location
d) Insert a new node at the end.
e) Display the contents of the list



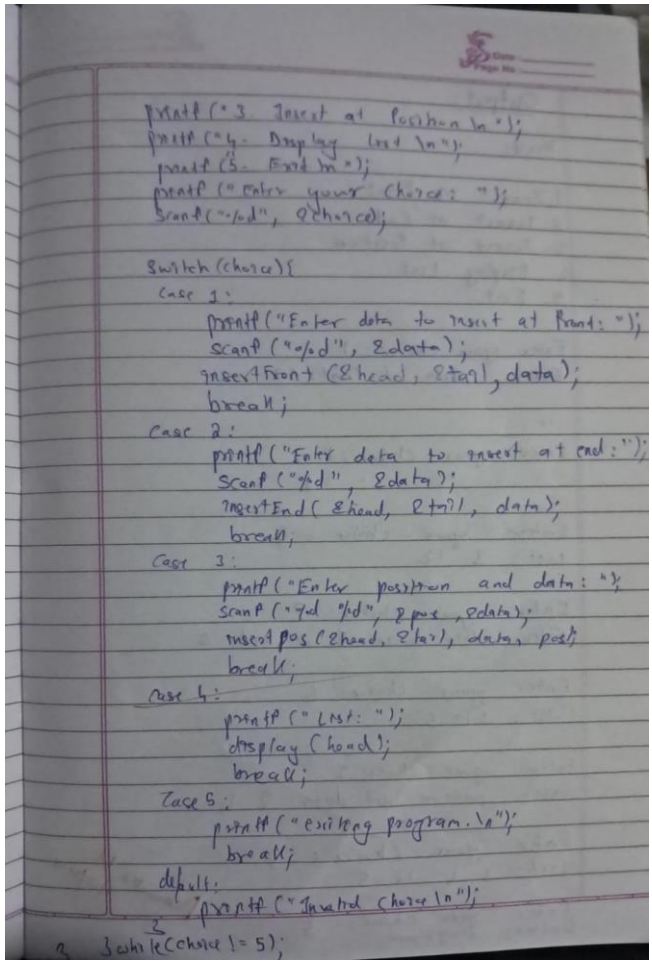
```
Week - 9

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

Node * Createnode (int x)
{
    Node* new = (Node*) malloc (sizeof (Node));
    new -> data = x;
    new -> next = NULL;
    new -> prev = NULL;
}

void insertfront (Node ** head, Node ** tail, int x)
{
    Node* temp = createnode(x);
    if (*head == NULL)
    {
        *head = temp;
        *tail = temp;
        return;
    }
    temp -> next = *head;
    (*head) -> prev = temp;
    *head = temp;
}
```



```
void insertend (Node** head, Node** tail, int x)
{
    Node* temp = Createnode(x);
    if (*head == NULL)
    {
        *head = temp;
        *tail = temp;
        return;
    }
    temp -> prev = *tail;
    (*tail) -> next = temp;
    *tail = temp;
}

void insertpos (Node** head, Node** tail, int data,
                int pos)
{ Node* temp = Create Node (data);
  if (pos == 1) {
      insertfront (head, tail, data);
      return;
  }
  Node* ptr = *head;
  int count = 1;
  while (ptr != NULL && count < pos - 1)
  { ptr = ptr -> next;
    count ++;
  }
  if (ptr == NULL){
      printf ("position out of range- \n");
  } else {
      temp -> next = ptr -> next;
      temp -> prev = ptr;
```



```
if (ptr->next != NULL)
{
    ptr->next->prev = temp;
}
else {
    *tail = temp;
}
ptr->next = temp;
}

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

void main() {
    Node* head = NULL;
    Node* tail = NULL;
    int choice, data, pos;

    do {
        printf("\n Menu:\n");
        printf("1. Insert at Front\n");
        printf("2. Insert at End\n");
```



```
printf("3. Insert at Position\n");
printf("4. Display List\n");
printf("5. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch(choice){
case 1:
    printf("Enter data to insert at Front: ");
    scanf("%d", &data);
    insertFront(&head, &tail, data);
    break;
case 2:
    printf("Enter data to insert at end: ");
    scanf("%d", &data);
    insertEnd(&head, &tail, data);
    break;
case 3:
    printf("Enter position and data: ");
    scanf("%d %d", &pos, &data);
    insertPos(&head, &tail, data, pos);
    break;
case 4:
    printf("List: ");
    display(head);
    break;
case 5:
    printf("exiting program.\n");
    break;
default:
    printf("Invalid choice\n");
}
} while(choice != 5);
```

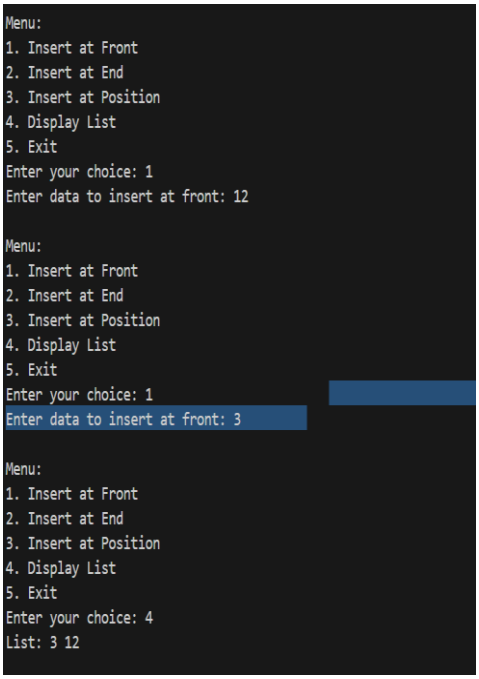
Code:

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


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


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


void insertFront(Node** head, Node** tail, int data) {
    Node* temp = createNode(data);
    if (*head == NULL) {
```

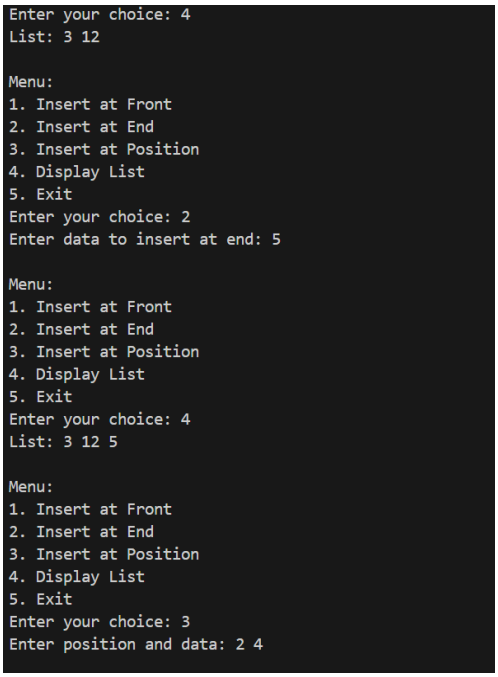
```
        *head = *tail = temp;
    } else {
        temp->next = *head;
        (*head)->prev = temp;
        *head = temp;
    }
}


void insertEnd(Node** head, Node** tail, int data) {
    Node* temp = createNode(data);
    if (*tail == NULL) {
        *head = *tail = temp;
    } else {
        (*tail)->next = temp;
        temp->prev = *tail;
        *tail = temp;
    }
}


void insertAtPos(Node** head, Node** tail, int data, int pos) {
    Node* temp = createNode(data);


    if (pos == 1) {
        insertFront(head, tail, data);
        return;
    }

    Node* ptr = *head;
    int count = 1;
    while (ptr != NULL && count < pos - 1) {
        ptr = ptr->next;
        count++;
    }

    if (ptr == NULL) {
        printf("Position out of range.\n");
    } else {
        temp->next = ptr->next;
        temp->prev = ptr;

        if (ptr->next != NULL) {
            ptr->next->prev = temp;
        } else {
            *tail = temp;
        }
        ptr->next = temp;
```

```
    }
}


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


int main() {
    Node* head = NULL;
    Node* tail = NULL;
    int choice, data, pos;

    do {
        printf("\nMenu:\n");
        printf("1. Insert at Front\n");
        printf("2. Insert at End\n");
        printf("3. Insert at Position\n");
        printf("4. Display List\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to insert at front: ");
                scanf("%d", &data);
                insertFront(&head, &tail, data);
                break;
            case 2:
                printf("Enter data to insert at end: ");
                scanf("%d", &data);
                insertEnd(&head, &tail, data);
                break;
            case 3:
                printf("Enter position and data: ");
                scanf("%d %d", &pos, &data);
                insertAtPos(&head, &tail, data, pos);
                break;
            case 4:
                printf("List: ");
                printList(head);
                break;
```

```
            case 5:
                printf("Exiting program.\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 5);

    return 0;
}
```



Output 1



Output 2

Output 3