VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

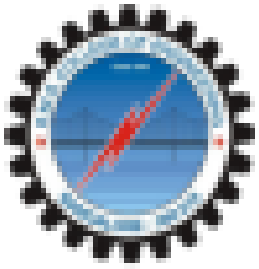
Data Structures using C Lab

(23CS3PCDST)

Submitted by

Shashank U (1BM23CS314)

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 **Shashank U(1BM23CS314),** 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 Namratha M Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

Index

Github Link: https://github.com/Shashank-u803/DSA-in-C/tree/main

Program 1

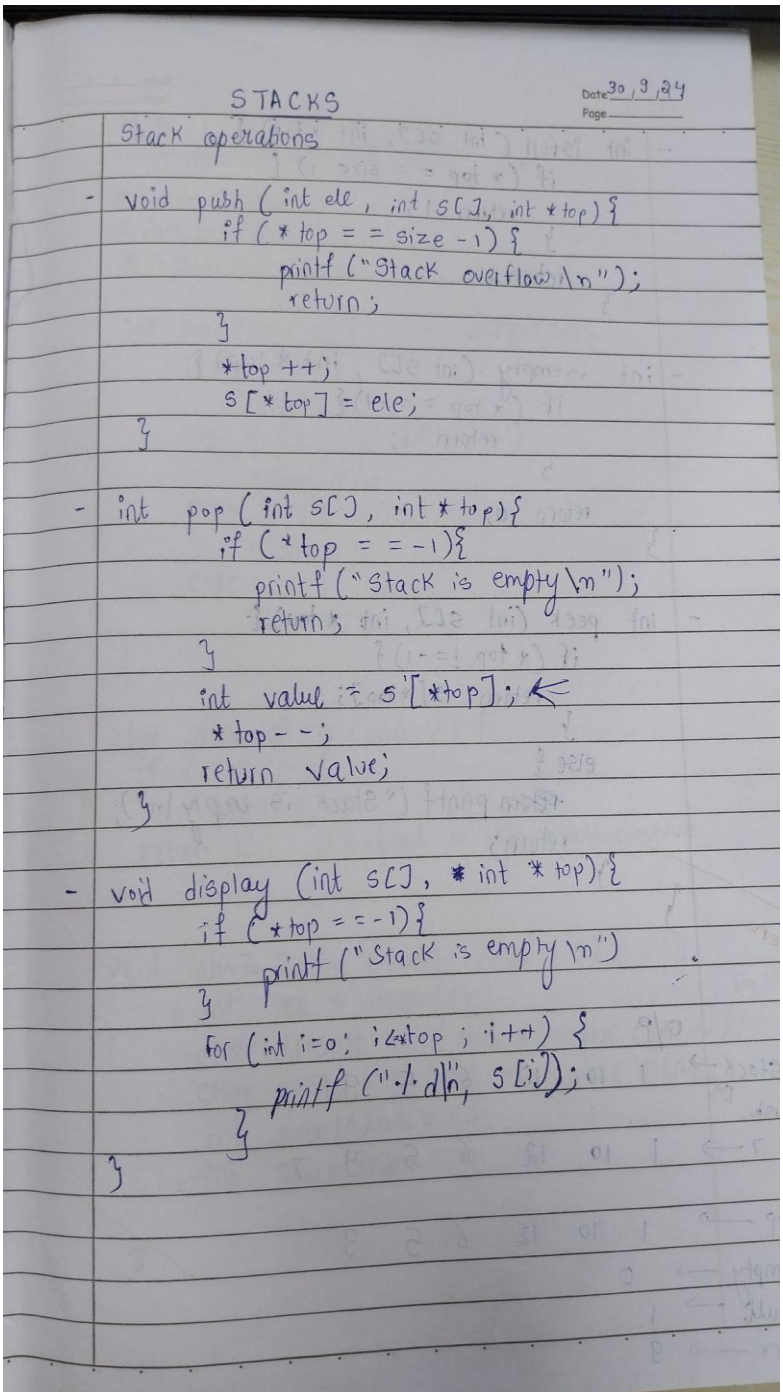
Write a program to simulate the working of stack using an array with the following:

a) Push

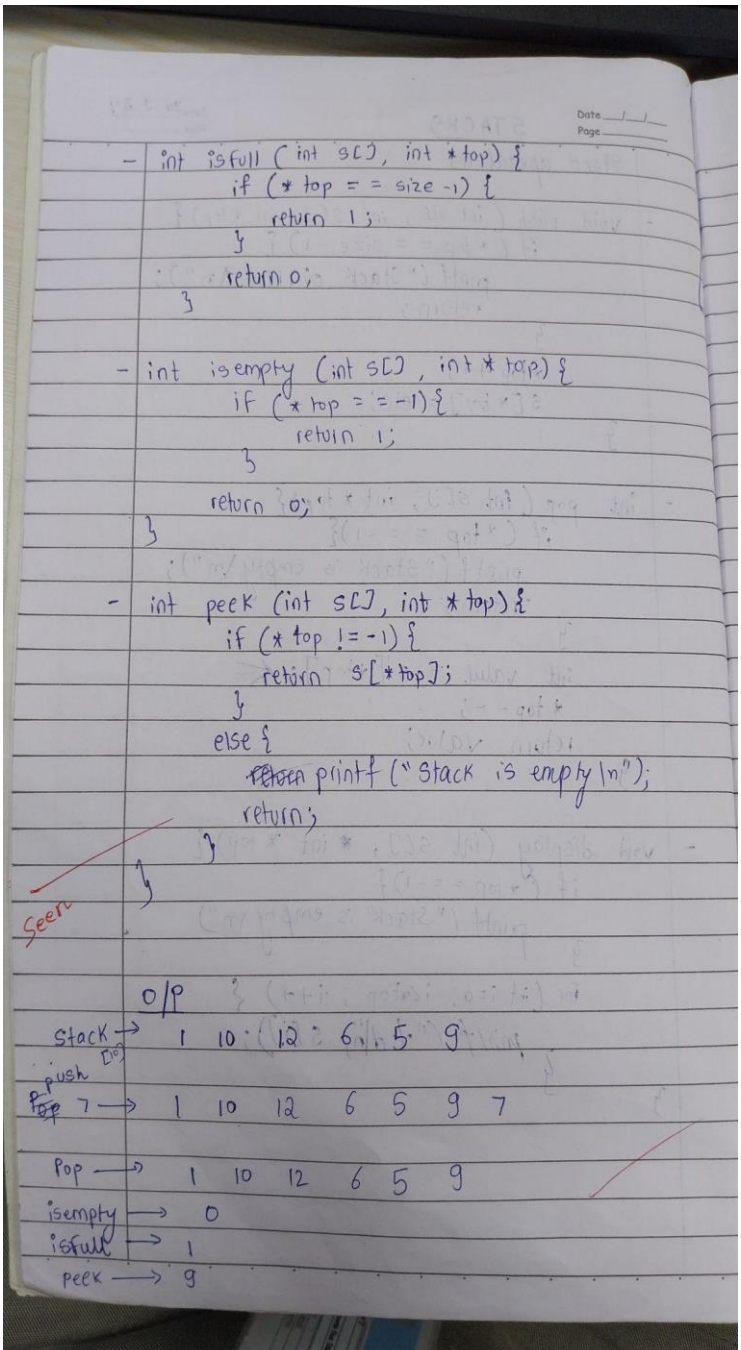
b) Pop

c) Display

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



```
STACKS                                   Date 30/9/24
Stack operations

- void push (int ele, int s[], int *top) {
      if (*top == size-1) {
          printf ("Stack overflow \n");
          return;
      }
      *top ++;
      s[*top] = ele;
  }

- int pop (int s[], int *top) {
      if (*top == -1) {
          printf ("Stack is empty \n");
          return;
      }
      int value = s[*top];
      *top--;
      return value;
  }

- void display (int s[], * int *top) {
      if (*top == -1) {
          print ("Stack is empty \n")
      }
      for (int i=0; i<=top; i++) {
          printf ("%d\n", s[i]);
      }
  }
```



```
- int isfull (int s[], int *top) {
      if (*top == size-1) {
          return 1;
      }
      return 0;
  }

- int isempty (int s[], int *top) {
      if (*top == -1) {
          return 1;
      }
      return 0;
  }

- int peek (int s[], int *top) {
      if (*top != -1) {
          return s[*top];
      }
      else {
          printf ("Stack is empty \n");
          return;
      }
  }
```

Seen

O/P

Stack → 1 10 12 6 5 9

push

top 7 → 1 10 12 6 5 9 7

Pop → 1 10 12 6 5 9

isempty → 0

isfull → 1

peek → 9

Code:

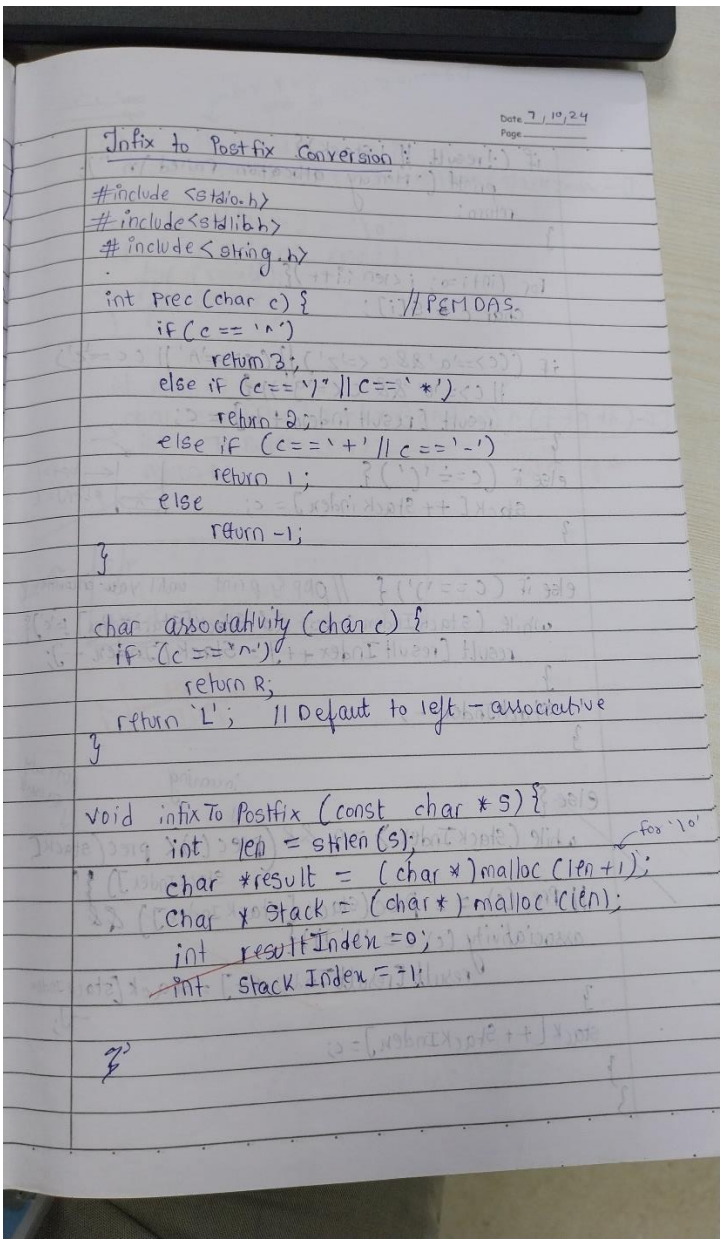
```
#include <stdio.h>
#define MAX 5

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");
        }
    }
}
```

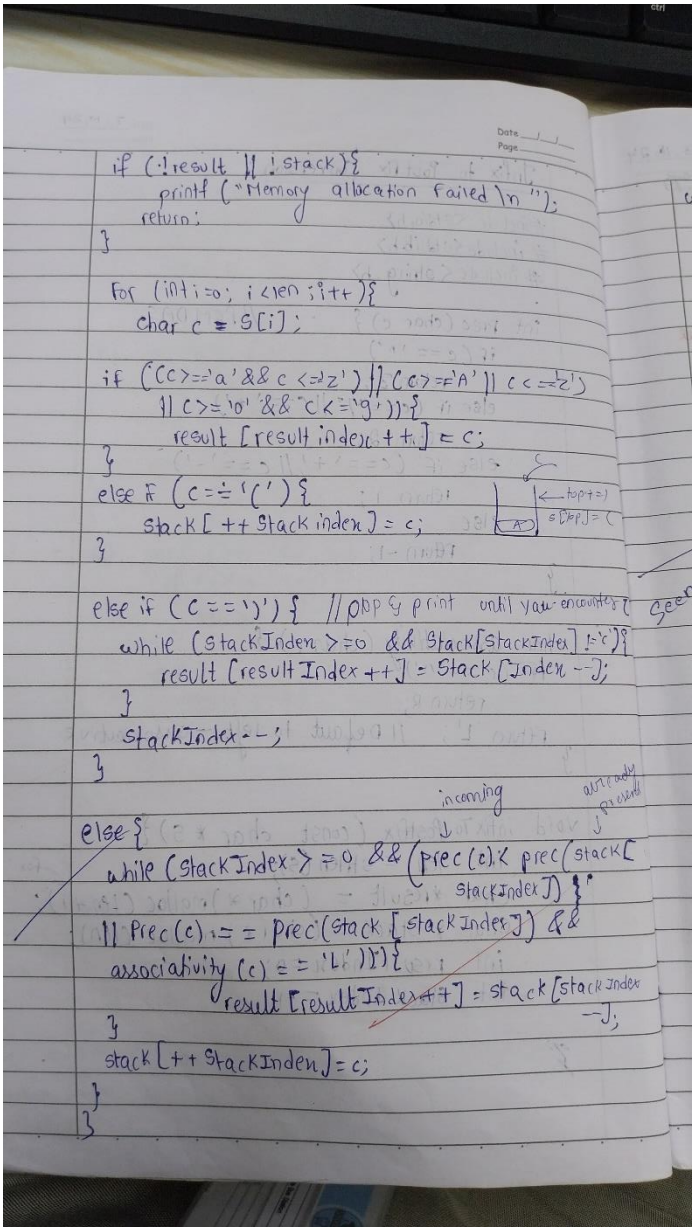
OUTPUT :

```
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>
```

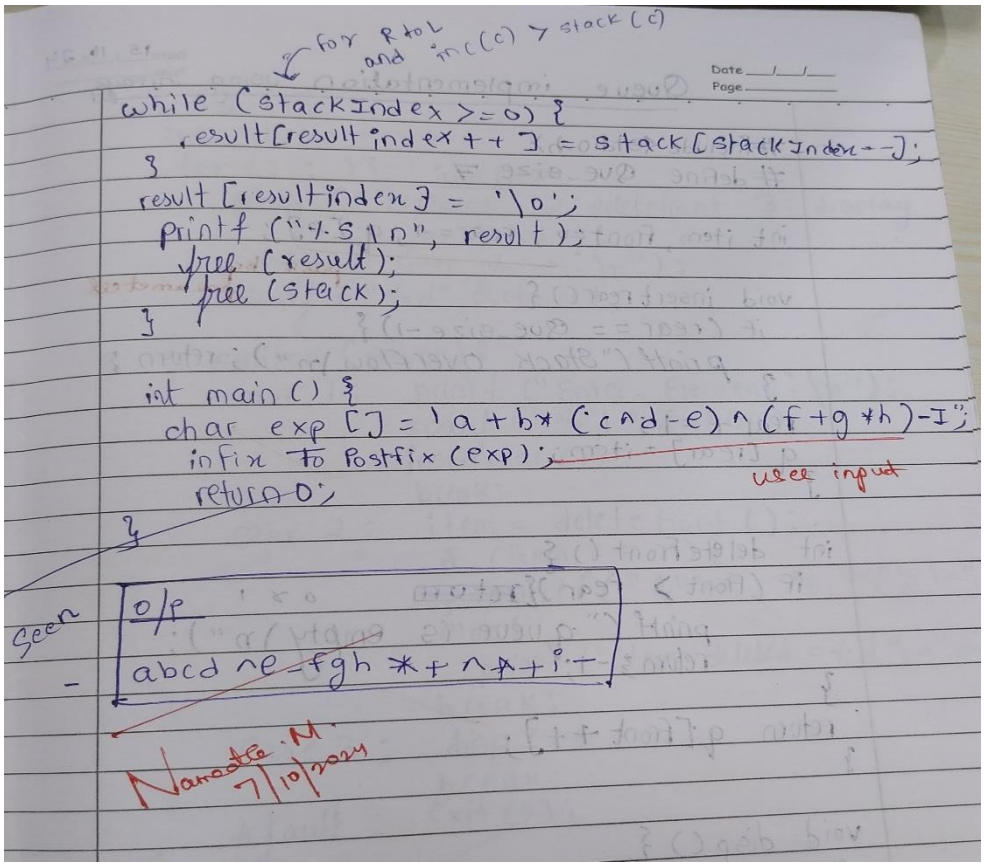
Program 2:
WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression.
The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide)



```
Infix to Postfix Conversion :

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

int Prec(char c) {          // PEMDAS.
    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';  // Default to left - associative
}

void infixToPostfix (const char *s){
    int len = strlen(s);
    char *result = (char*)malloc(len+1);
    Char * stack = (char*) malloc(len);
    int resultIndex = 0;
    int stackIndex = -1;
}
```



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

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

if ((c>='a' && c<='z') || (c>='A' || c<='Z')
    || c>='0' && c<='9')){
    result[resultIndex++] = c;
}
else if (c=='('){
    stack[++stackIndex] = c;
}

else if (c==')'){  // pop & print until you encounters (
    while (stackIndex>=0 && stack[stackIndex]!='('){
        result[resultIndex++] = stack[index--];
    }
    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;
}
}
```



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
```

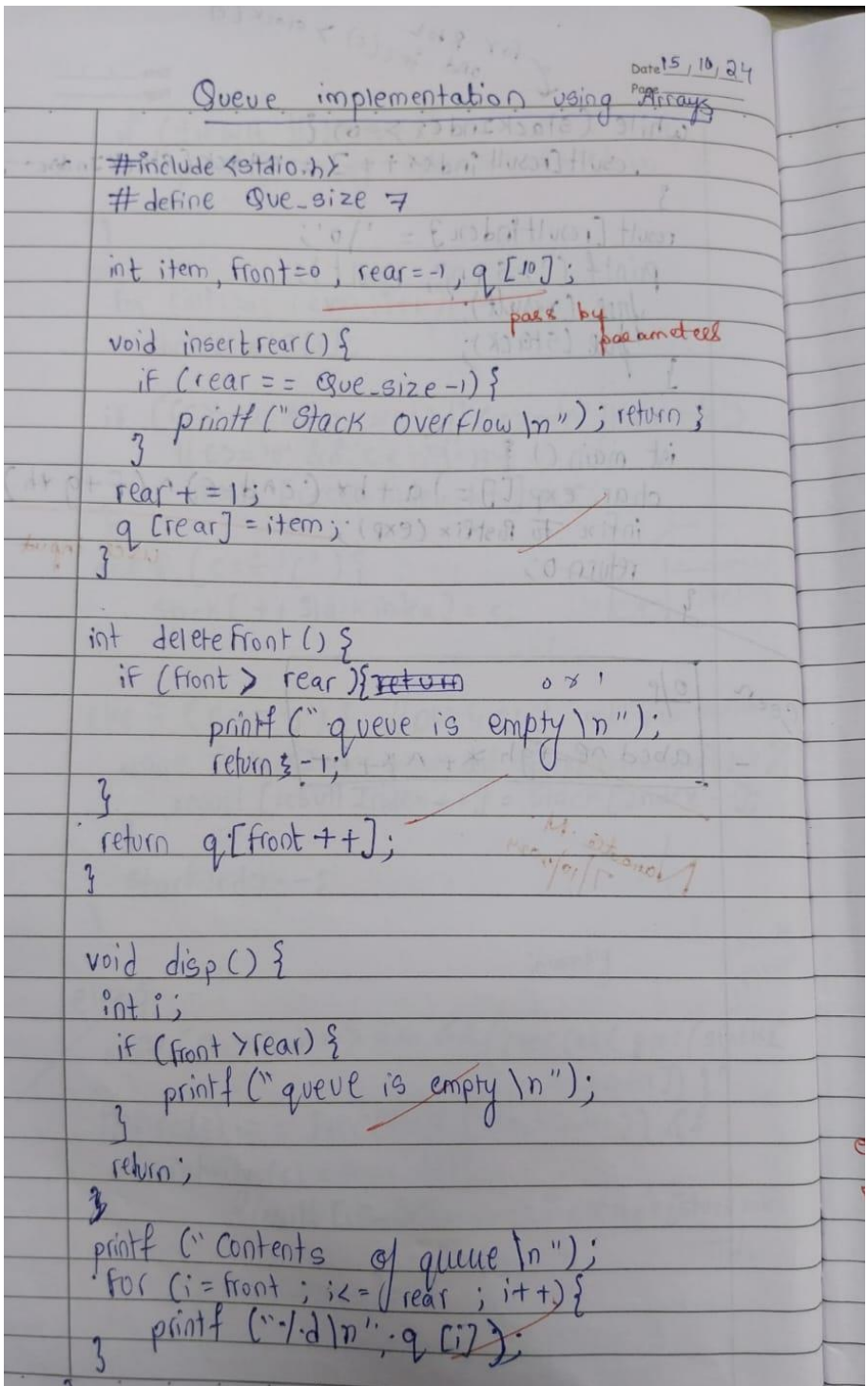
```
        return -1;
 }



// 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;
}
```

```
PS C:\Users\Admin\Desktop\1BM23CS314> cd c
PS C:\Users\Admin\Desktop\1BM23CS314\c> gcc .\Queue.c
PS C:\Users\Admin\Desktop\1BM23CS314\c> .\a.exe
Enter an infix expression: a+b*c-q/b+p^g
```

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



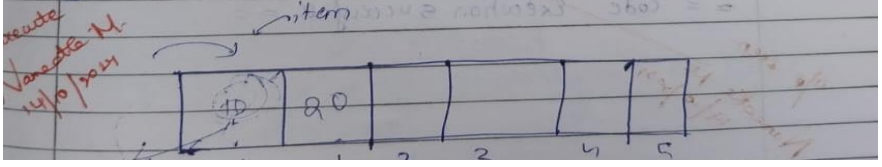
Queue implementation using Arrays

Date 15, 10, 24
Page

```
#include <stdio.h>
#define Que_size 7

int item, front=0, rear=-1, q[10];

void insertrear(){
  if (rear == Que_size-1){
    printf("Stack Overflow\n"); return;
  }
  rear+=1;
  q[rear] = item;
}

int deleteFront(){
  if (front > rear){
      printf("queue is empty\n");
      return -1;
  }
  return q[front++];
}

void disp(){
  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]);
}
```

pass by parameters

```
int main(){
  int choice;
  for(;;){                    insertrear
    printf("1:choice 2:deletefront 3: display
            4: exit \n");
    printf("Enter choice:\n");
    scanf("%d",&choice);
  switch(choice){
    case 1: printf("Enter Element:\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: disp();
            break;
    default: exit(0);
  }
}
return 0;
}
```



o/p

```
1: Insert  2: Delete  3: Display  Any other: exit
1
Enter item to insert:
25
25 inserted successfully

1: Insert  2: Delete  3: Display  Any other: exit
1
Enter item to ~~del~~ insert:
35
35 inserted successfully

1: Insert  2: Delete  3: Display  Any other: exit
2
25 deleted successfully

1: Insert  2: Delete  3: Display  Any other: exit
3
35

1: Insert  2: Delete  3: Display  Any other: exit
7

== code Execution successfull ==
```

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;
}
```

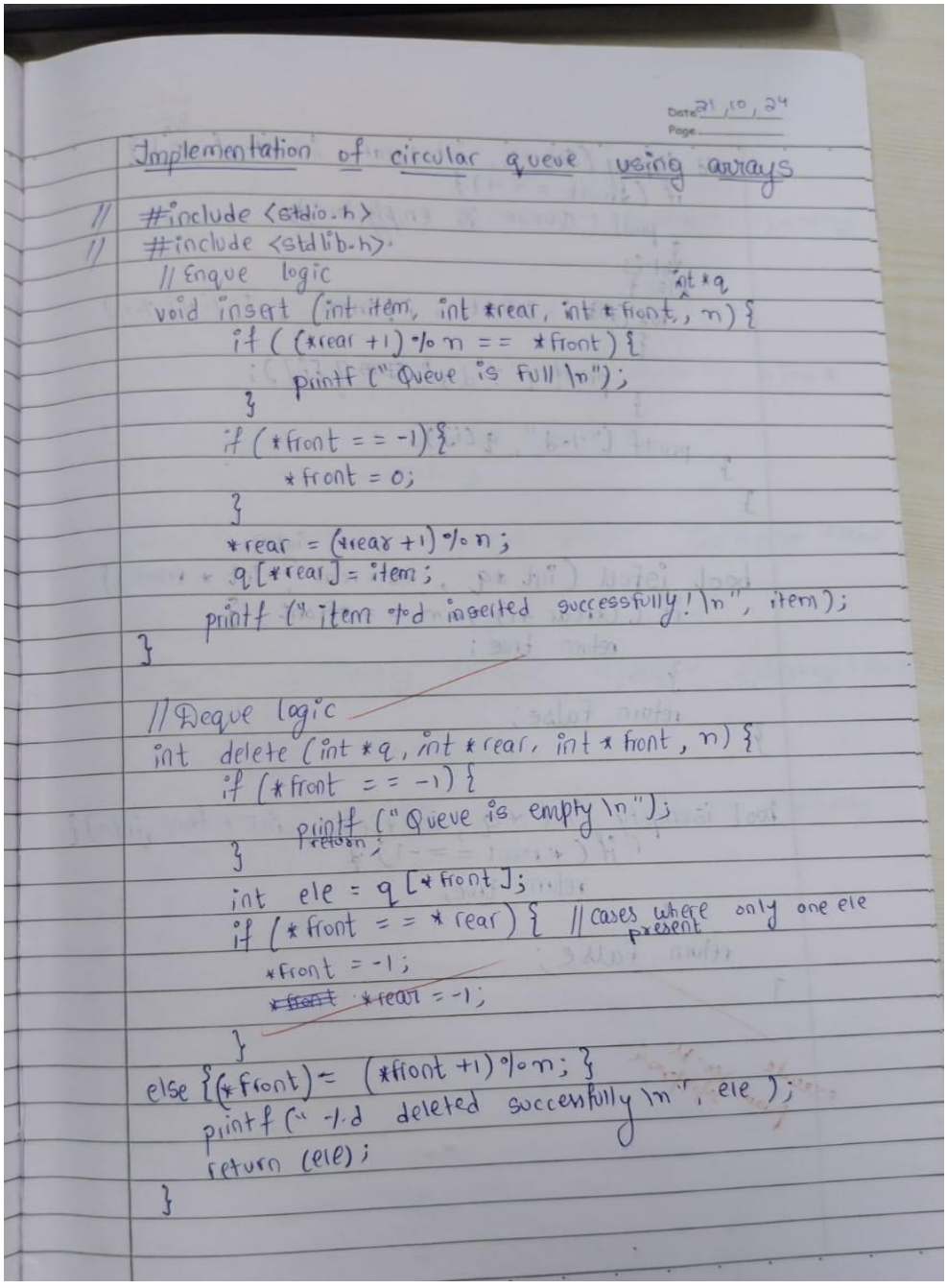
Output:

```
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

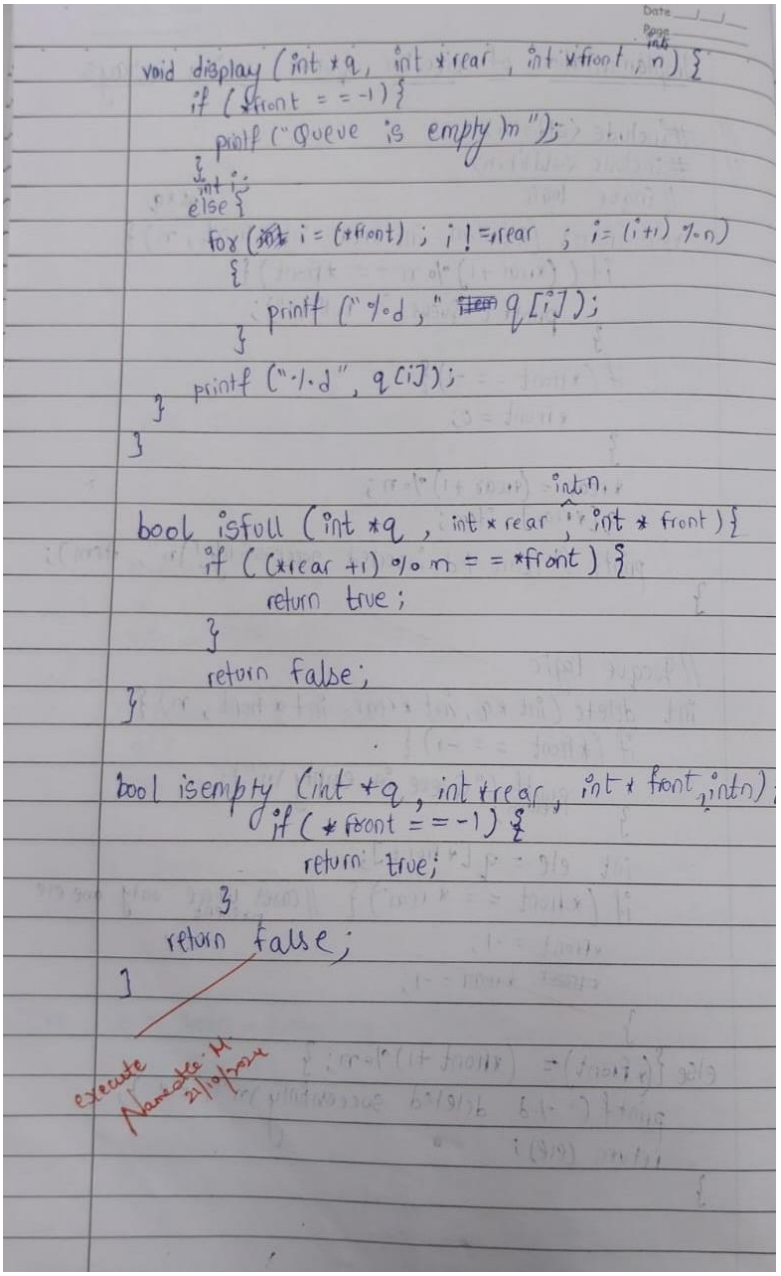
The program should print appropriate messages for queue empty and queue overflow conditions



Date 21, 10, 24
Page

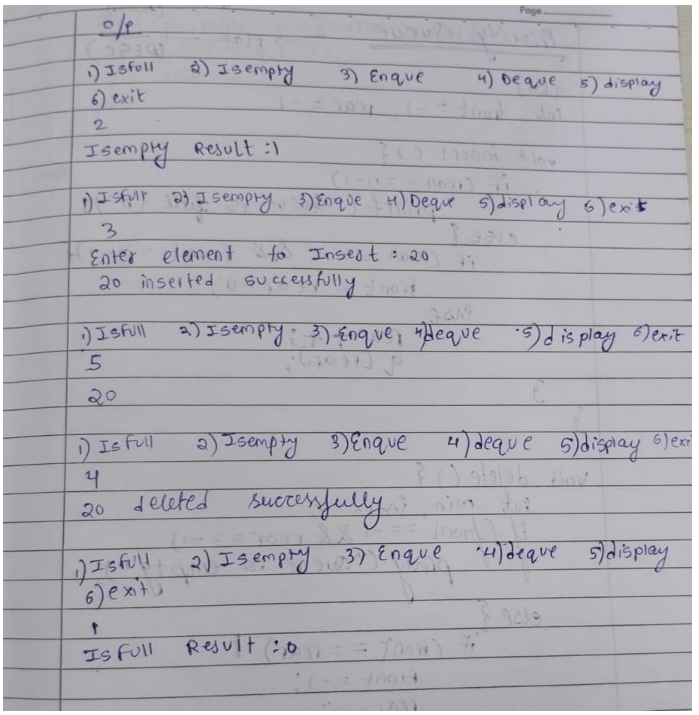
Implementation of circular queue using arrays

```
// #include <stdio.h>
// #include <stdlib.h>
// Enque logic
void insert (int item, int *rear, int *front, int *q, n) {
    if ((*rear +1) % n == *front) {
        printf ("Queue is full \n");
    }
    if (*front == -1) {
        *front = 0;
    }
    *rear = (*rear +1) % n;
    q[*rear] = item;
    printf ("item %d inserted successfully! \n", item);
}

// Deque logic
int delete (int *q, int *rear, int *front, n) {
    if (*front == -1) {
        printf ("Queue is empty \n");
        return;
    }
    int ele = q[*front];
    if (*front == *rear) { // cases where only one ele present
        *front = -1;
        *rear = -1;
    }
else {(*front) = (*front +1) % n; }
    printf ("%d deleted successfully \n", ele);
    return (ele);
}
```



```
void display (int *q, int *rear, int *front, int n) {
    if (*front == -1) {
        printf ("Queue is empty \n");
    }
    int i;
    else {
        for (i = (*front) ; i != *rear ; i = (i+1) % n)
        {
            printf ("%d ", q[i]);
        }
        printf ("%d", q[i]);
    }
}

bool isfull (int *q, int *rear, int *front, int n) {
    if ((*rear +1) % n == *front) {
        return true;
    }
    return false;
}

bool isempty (int *q, int *rear, int *front, int n);
    if (*front == -1) {
        return true;
    }
    return false;
}
```

execute
Namrata M
21/10/2022



```
o/p
1) Isfull  2) Isempty  3) Enque  4) Deque  5) display
6) exit
2
Isempty Result : 1

1) Isfull  2) Isempty  3) Enque  4) Deque  5) display  6) exit
3
Enter element to Insert : 20
20 inserted successfully

1) Isfull  2) Isempty  3) Enque  4) Deque  5) display  6) exit
5
20

1) Isfull  2) Isempty  3) Enque  4) deque  5) display  6) exit
4
20 deleted successfully

1) Isfull  2) Isempty  3) Enque  4) deque  5) display
6) exit
1
Is Full  Result : 0
```

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:

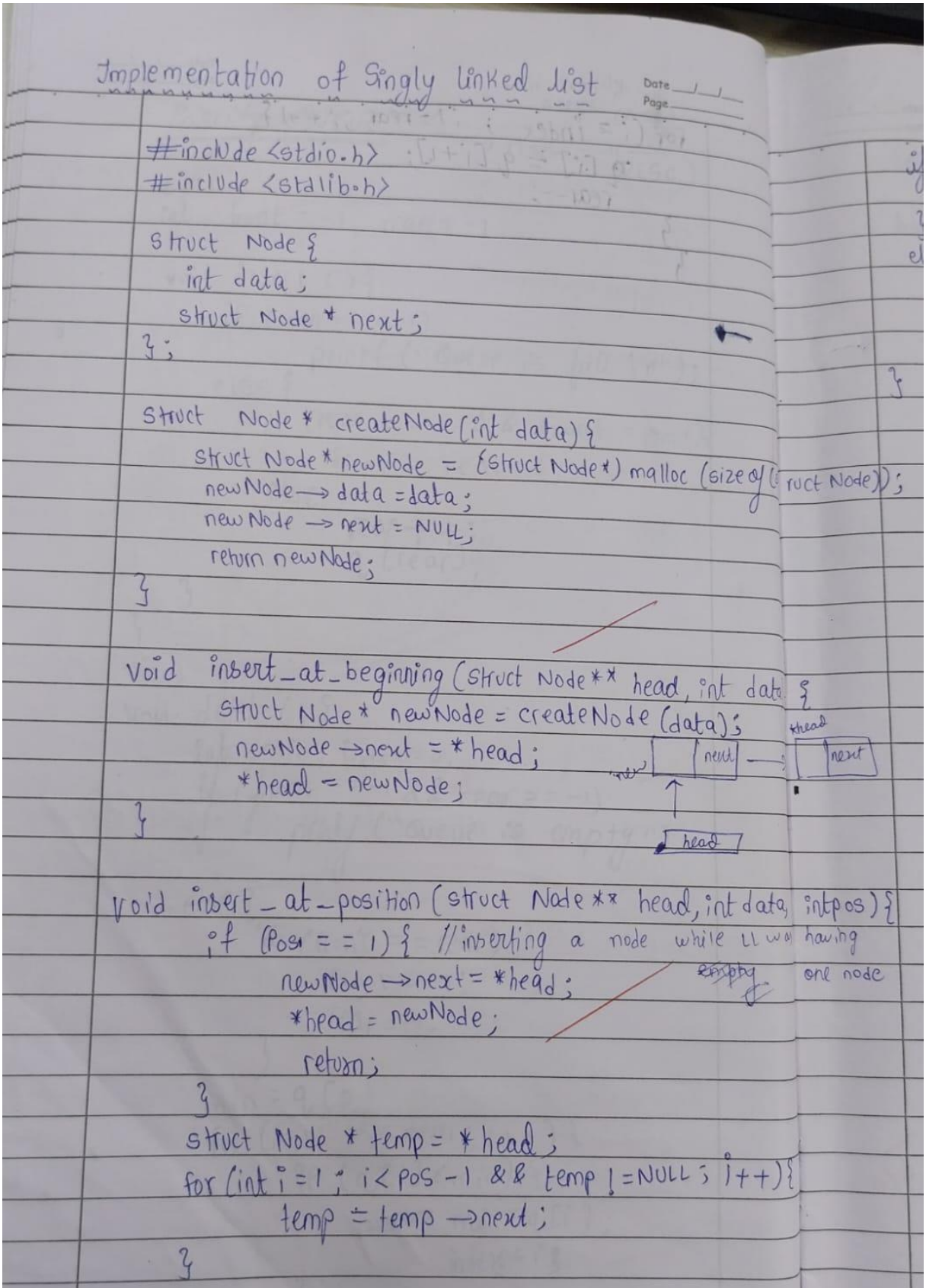
05(a) WAP to Implement Singly Linked List with following operations

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.

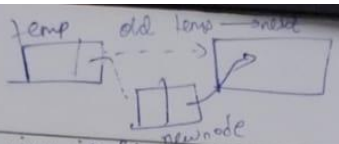
5b) Leetcode problem no.20 (Valid parantheses)



Implementation of Singly Linked List

Date / /
Page

```
#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 insert_at_beginning (struct Node** head, int data) {
    struct Node* newNode = createNode (data);
    newNode -> next = *head;
    *head = newNode;
}

void insert_at_position (struct Node** head, int data, int pos) {
    if (Pos == 1) { // inserting a node while LL was having one node
        newNode -> next = *head;
        *head = newNode;
        return;
    }
    struct Node * temp = *head;
    for (int i = 1; i < pos - 1 && temp != NULL; i++) {
        temp = temp -> next;
    }
```



Date __/__/__
Page

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

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

void display (struct Node * head) {
    if (head == NULL) {
        printf ("Empty list!\n");
        return;
    }
    struct Node * temp = head; // traverse from beginning
    while (temp != NULL) {
        printf ("%d -> ", temp -> data);
        temp = temp -> next;
    }
    printf ("Null\n");
}
```

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

    insert_at_beginning (&head, 5);
    display (head);

    insert_at_pos (&head, 15, 2);
    display (head);

    insert_at_end (&head, 30);
    display (head);

    return 0;
}
```

execute Namedee. M 29/10/24

O/P

NULL
5 → NULL
5 → 15 → NULL
5 → 15 → 30 → NULL

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.

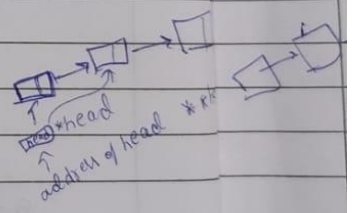
c) Display the contents of the linked list.

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

Date 17/11/24
Page

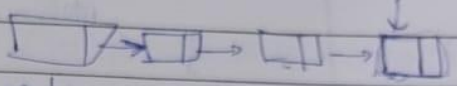
Singly linked list : Deletion at beginning, end, specified position.

```
#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 deleteFirst (struct Node ** head ) {
    if (*head == NULL) {
        printf ("List is empty \n");
        return;
    }
    struct Node * temp = *head ;
    *head = (*head) -> next;
    free (temp);
}

void deleteEnd (struct Node **head) {
    if (*head == NULL) {
        printf ("Linked list is Empty. \n");
        return;
    }
```

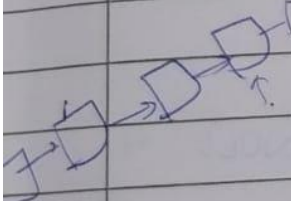


Date __/__/__
Page _____

```
if ((*head) -> next == NULL) {   // only one node
    Free (*head);
    *head = NULL;
    return;
}
```



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

void delete (struct Node **head, int value) {
    if (*head == NULL) {
        printf ("Linked list is empty\n");
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL && temp->next->data != value)
    {
        temp = temp->next;
    }
    if (temp->next == NULL) {
        print ("Node not found \n");
        return;
    struct Node* delNode = temp->next; // node to be deleted
    temp->next = temp->next->next;
    free (delNode);
}
```



Date / /
Page

```
void display (struct Node *head ) {
    if (head == NULL) {
        printf ("Linked list is empty \n");
        return;
    }
    struct Node* temp = head;
    printf ("Linked list : ");
    while (temp != NULL) {
        printf ("%d -> ", temp->data);
    }
    printf ("NULL \n");
}
```

Navneeta M.
11/11/2024

o/p

5 → NULL

5 → 10 → NULL

5 → 10 → 15 → NULL

5 → 10 → 15 → 20 → 25 → NULL

5 → 10 → 15 → 20 → 25 → 30 → NULL

delete first: ~~5~~ → 10 → 15 → 20 → 25 → 30 → NULL

delete last: 10 → 15 → 20 → 25 → NULL

delete is: 10 → 20 → 25 → NULL.

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>
```

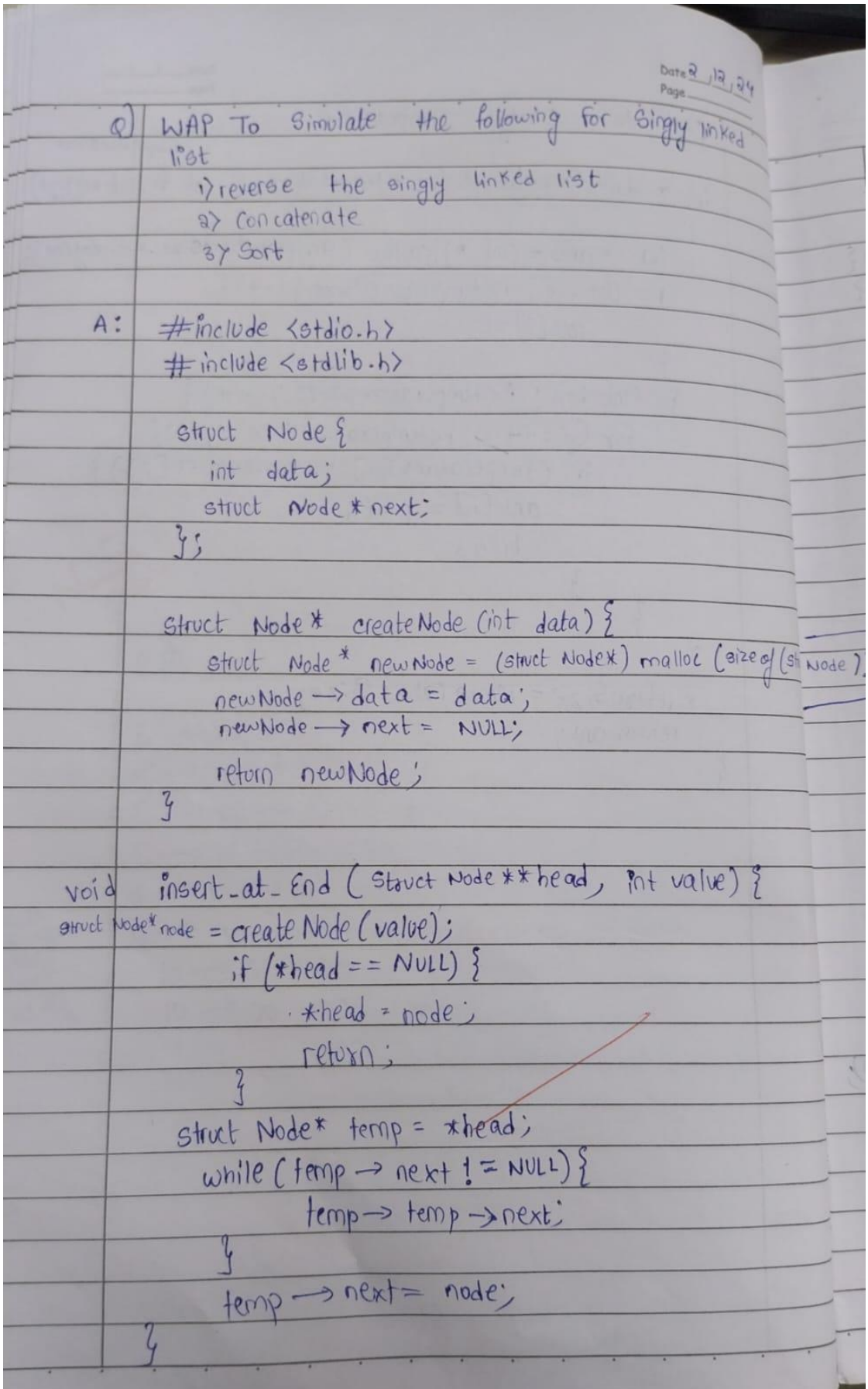
Leetcode problem(Daily Temperatures):

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:

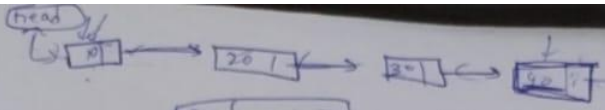
WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists.



Q) WAP To Simulate the following for Singly linked list
1) reverse the singly linked list
2) Concatenate
3) Sort

A:

```
#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(st Node));
    newNode -> data = data;
    newNode -> next = NULL;
    return newNode;
}

void insert_at_End (Struct Node **head, int value) {
struct Node* node = createNode (value);
    if (*head == NULL) {
        *head = node;
        return;
    }
struct Node* temp = *head;
while (temp -> next != NULL) {
    temp -> temp -> next;
}
temp -> next = node;
}
```



next → 20
curr → curr
Date / /
Project = null

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

```
void concatenate (struct Node *h1, struct Node*h2){
    struct Node* temp1 = h1; *temp2 = h2;
    while (temp1 -> next != NULL){
        temp1 = temp1 -> next;
    }
    temp1 -> next = temp2; // connect last node of 1st LL to 1st node of 2nd LL
    temp1 = head h1;
    while (temp1 -> next != NULL){
        temp1 = temp1 -> next;
        printf ("%d -> ", temp1 -> data;
        temp1 -> temp1 -> next;
    }
    printf ("NULL");
}
```

o/p

Sample : 10 → 12 → 15 → 17 → 19 → 21 → 16 LL

Sorting → 10 → 12 → 15 → 16 → 17 → 19 → 21

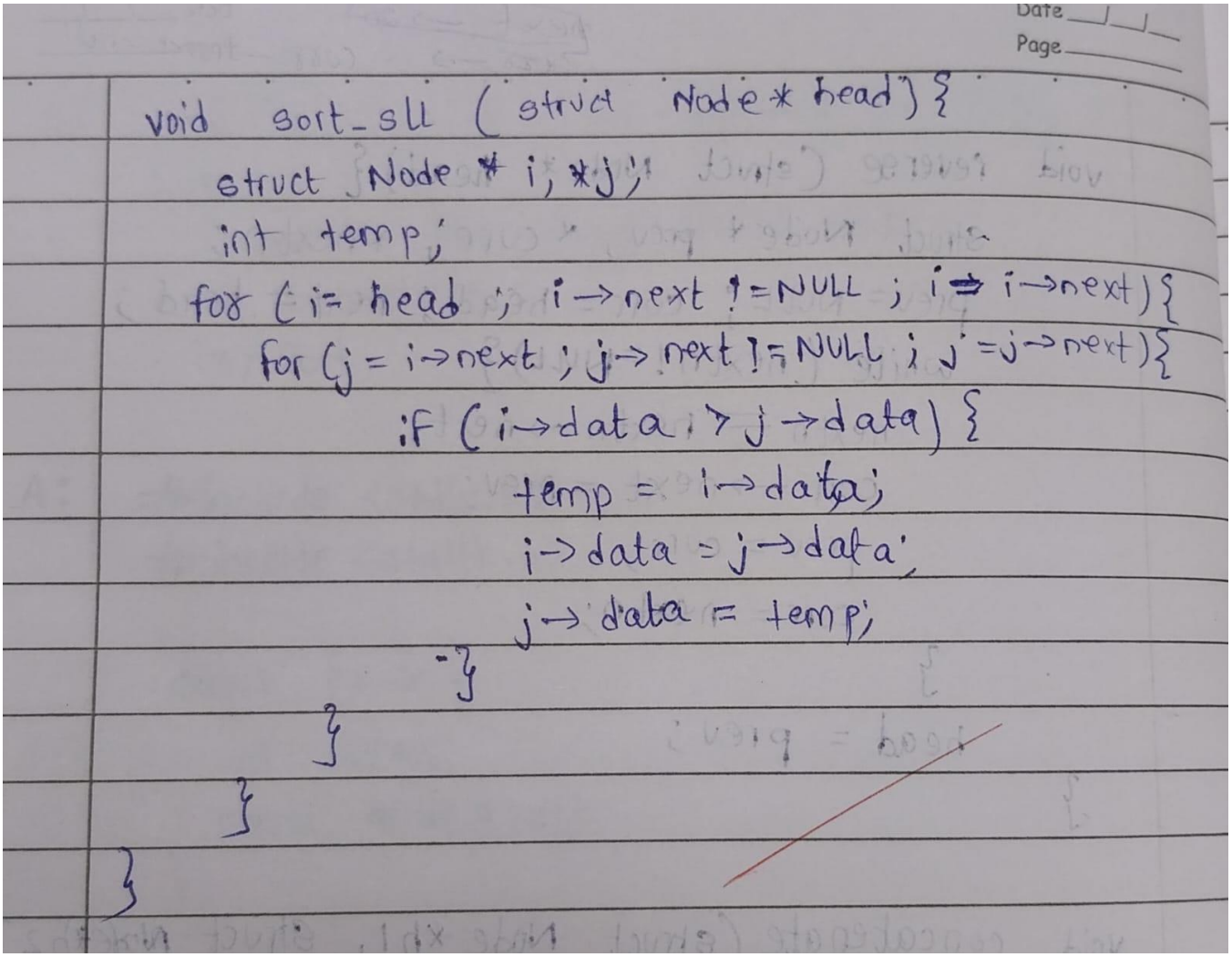
Reverse → 16 → 21 → 19 → 17 → 15 → 12 → 10

LL 2 → 5 → 10 → 12 → 60

Concatenate : 10 → 12 → 15 → 17 → 19 → 21 → 16 → 5 → 10 → 12 → 60

*ptr = 10;

*ptr =



Code:

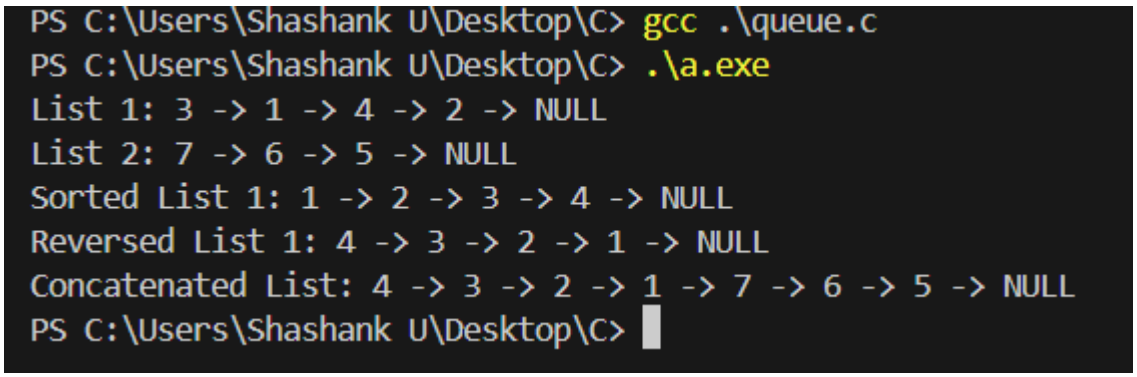
```
#include <stdio.h>
#include <stdlib.h>

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;
```

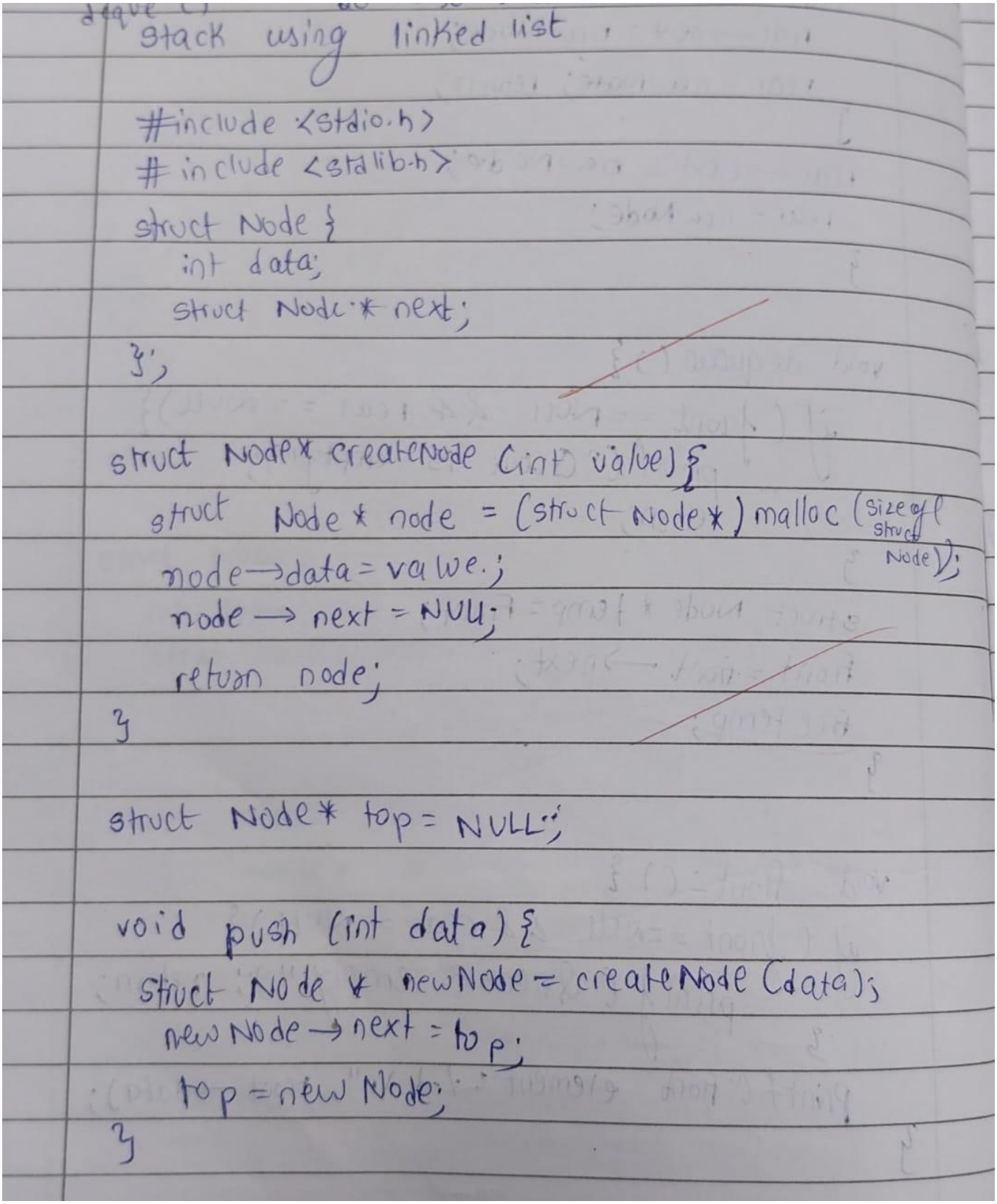
```
    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head = prev;
}

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:



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



```
stack using linked list

#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("Stack is empty\n");
        return;
    }
    for (struct Node *temp = top; temp != NULL; ) {
        printf("%d ", temp->data);
    }
    printf("\n");
}
```

Sample stack: 5 → 10 → 15 → 20 → 25 (top)

Push 12: 5 → 10 → 15 → 20 → 25 → 12 (top)

Pop: 5 → 10 → 15 → 20 → 25 (top)

Peek: 25

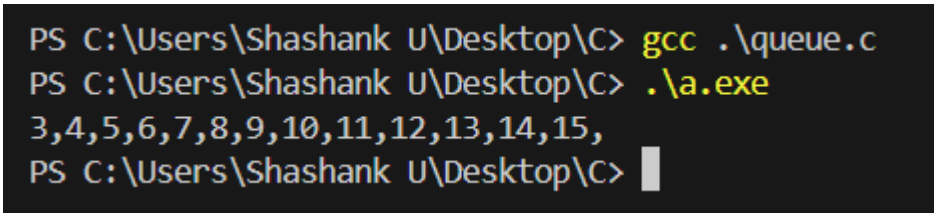
Display: 5 → 10 → 15 → 20 → 25

Code:

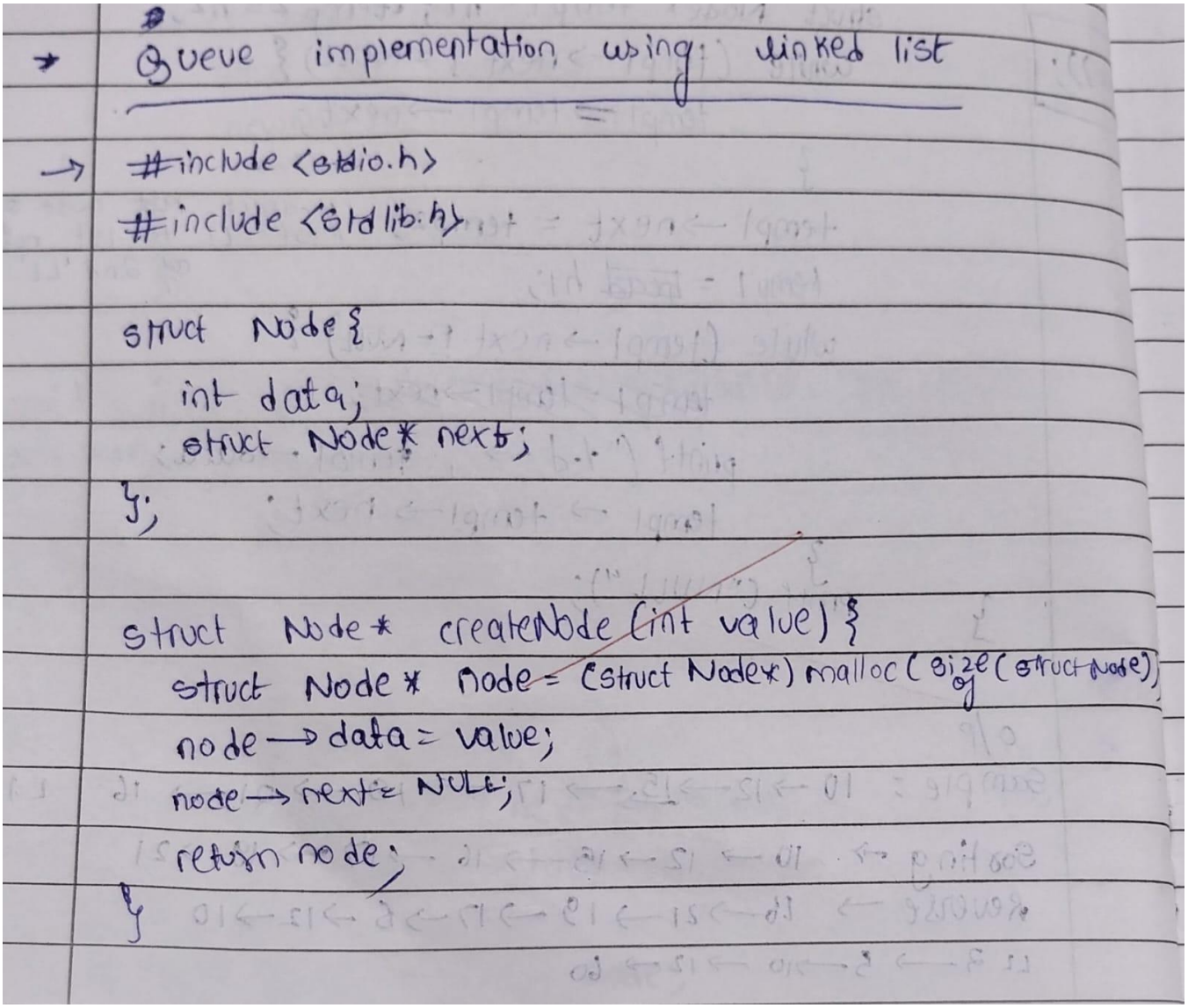
```
#include<stdio.h>
#include<stdlib.h>

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:



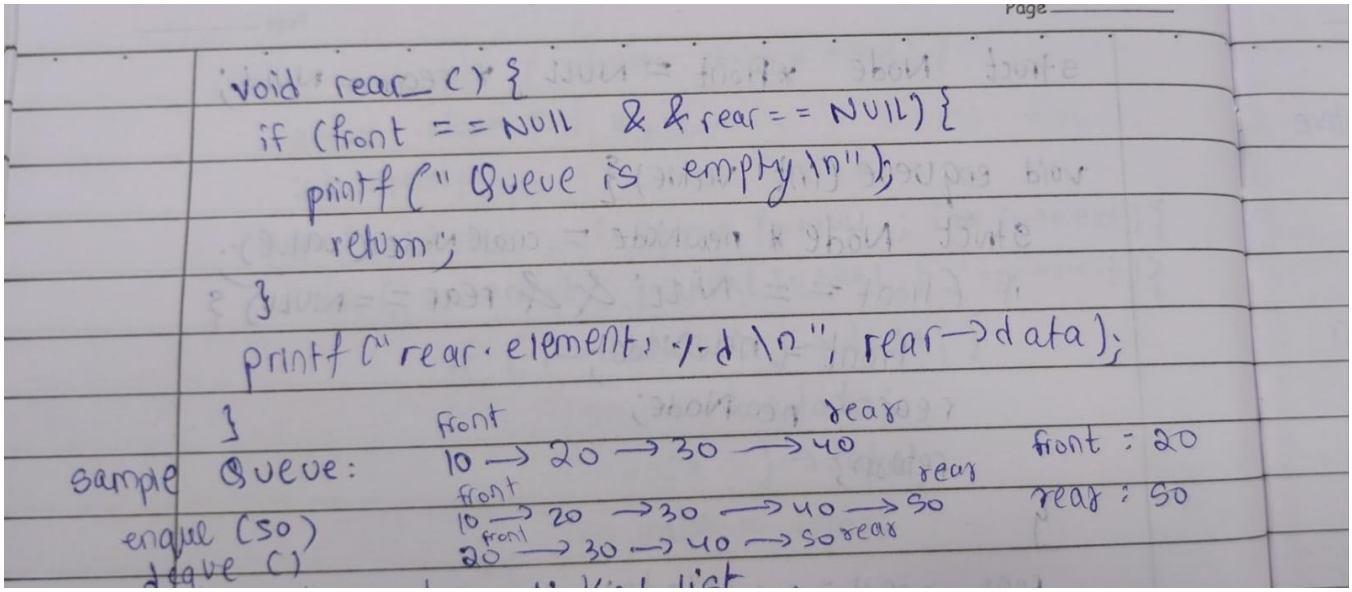
Problem 9:
WAP to Implement Single Link List to simulate Queue Operations.



Queue implementation using linked list

```
#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; return;
}
rear -> next = newNode;
rear = newNode;
}

void dequeue () {
    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);
}
```



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

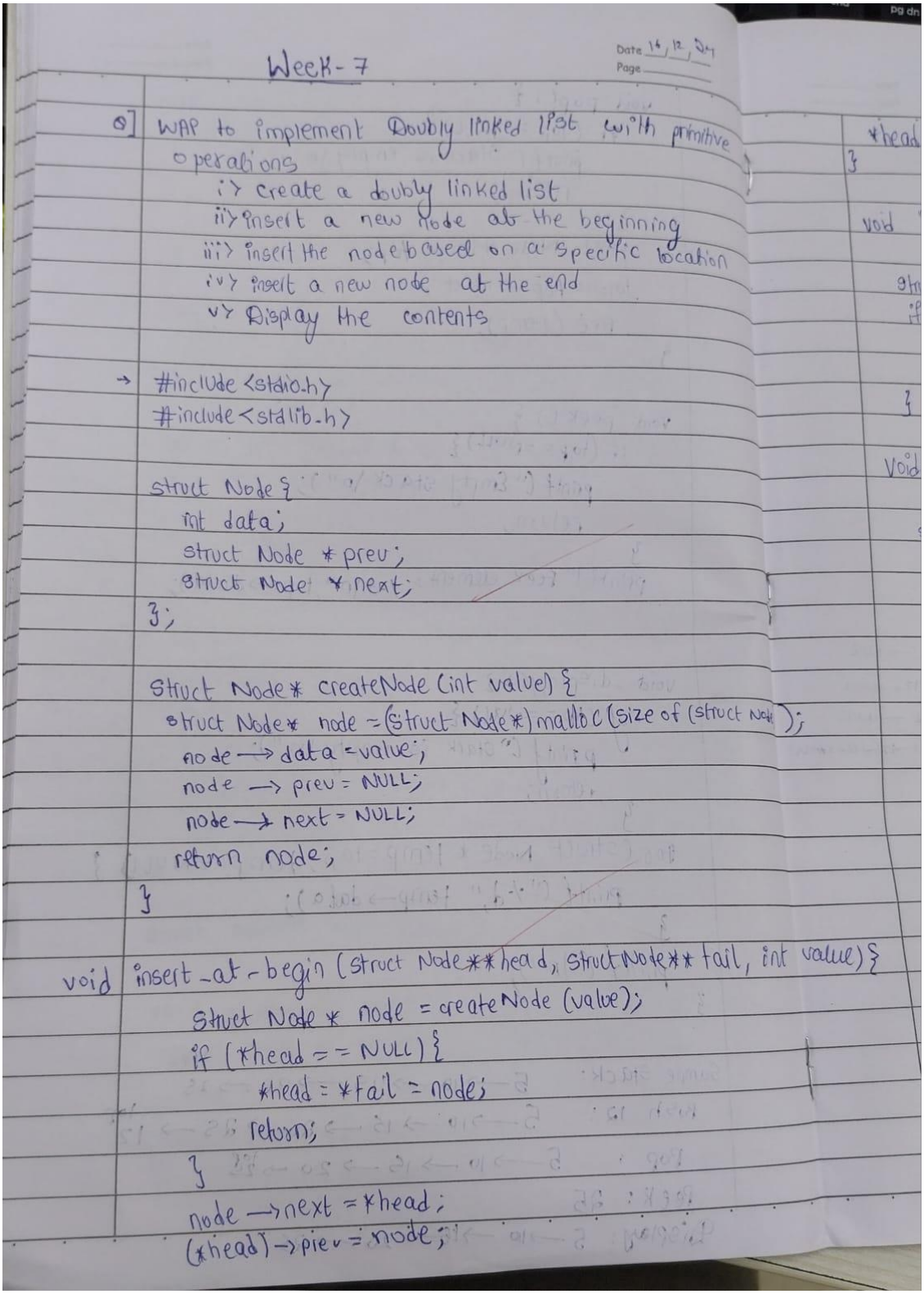
a) Create a doubly linked list.

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



Week-7

Date 14/12/24
Page

Q] WAP to implement Doubly linked list with primitive operations
i) create a doubly linked list
ii) Insert a new node at the beginning
iii) Insert the node based on a specific location
iv) insert a new node at the end
v) Display the contents

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

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

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

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
}
void
str
if
}
Void

```
    *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;
    }

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");
            return; free (node);
        }
        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 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");
}
```

Namaste M. 16/12/2024

O/P

Sample linked list : NULL <-> 10 <-> 15 <-> 11 <-> 17 <-> NULL

insert_at_end (12) NULL <-> 10 <-> 15 <-> 11 <-> 17 <-> 12 <-> NULL

insert_at_begin (8) NULL <-> 8 <-> 10 <-> 15 <-> 11 <-> 17 <-> 12 <-> NULL

insert_at_pos (2, 100) NULL <-> 8 <-> 100 <-> 10 <-> 15 <-> 11 <-> 17 <-> 12 <-> NULL

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;
}
```

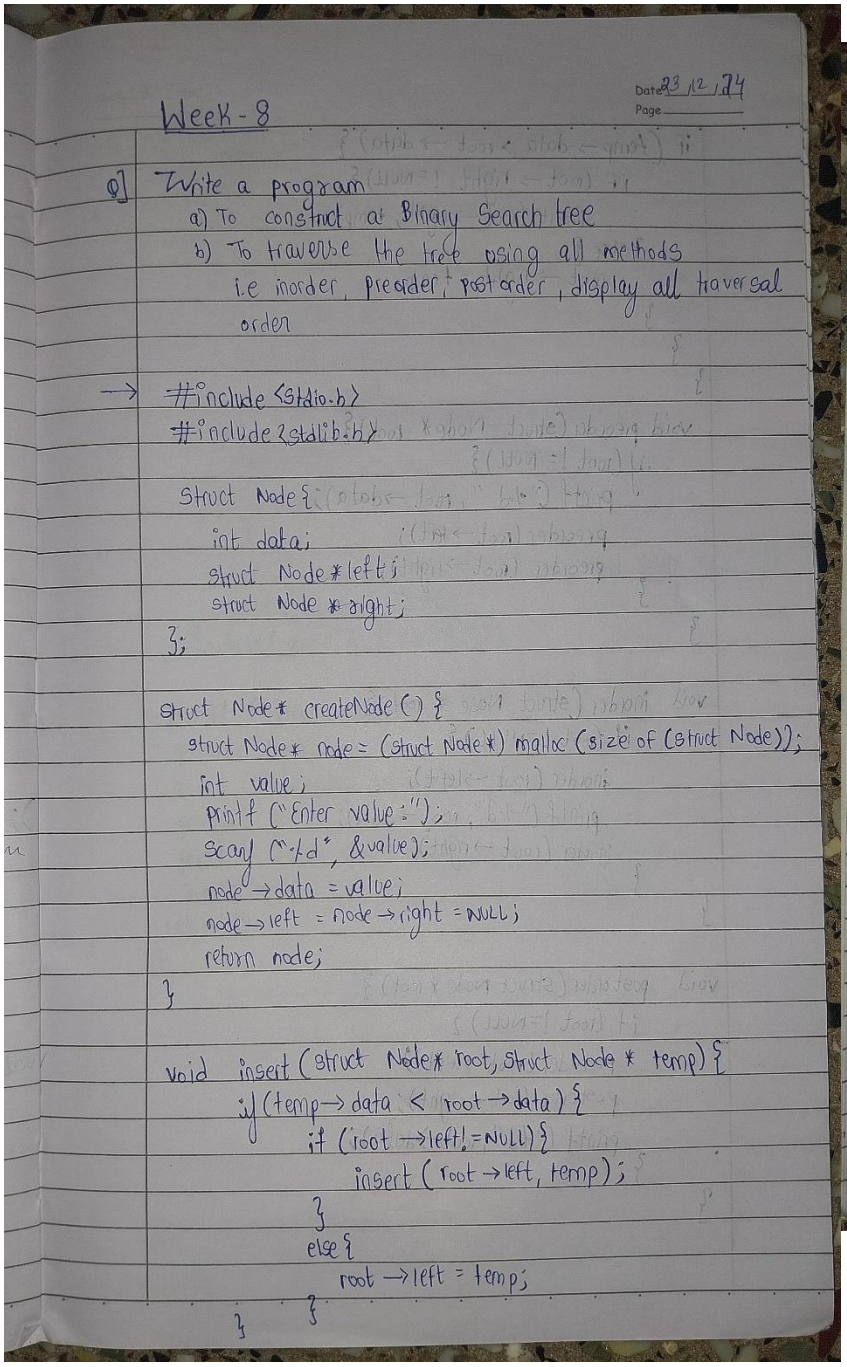
Output:

```
PS C:\Users\Shashank U\Desktop\C> gcc .\queue.c
PS C:\Users\Shashank U\Desktop\C> .\a.exe
Null->10->NULL(tail)
Null->5->10->NULL(tail)
Null->5->10->15->NULL(tail)
Null->5->10->15->20->25->30->35->40->NULL(tail)
Null->5->60->10->15->20->25->30->35->40->NULL(tail)
Position out of bounds
Null->5->60->10->15->20->25->30->35->40->NULL(tail)
Element 30 found in position 7
Element 100 not found
Null->5->60->10->15->20->25->30->35->40->NULL(tail)
NULL<-40<-35<-30<-25<-20<-15<-10<-60<-5<-NULL(head)
PS C:\Users\Shashank U\Desktop\C>
```

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



Week-8

Date 23/12/24

Q] Write a program
a) To construct a Binary Search tree
b) To traverse the tree using all methods
i.e inorder, preorder, postorder, display all traversal order

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

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

```
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 wanna continue? if yes then enter Y/y :");
     scanf (" %c", &ch);
    } while (ch =='Y' || ch =='y');

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

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;
}
```

```
PS C:\Users\Shashank U> cd desktop/c
PS C:\Users\Shashank U\desktop\c> gcc .\BST.c
PS C:\Users\Shashank U\desktop\c> .\a.exe
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
PS C:\Users\Shashank U\desktop\c>
```

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

Week-9

Date 23/12/24
Page

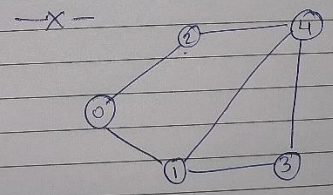
Q] Write a program to traverse a graph using BFS & DFS method.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX 20
void BFS(int adj[MAX][MAX], int v, int s) {
    int q[MAX], front=0, rear=0;
    bool visited[max] = {false};
    visited[s] = true;
    q[rear++] = s;
    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[MAX][MAX], int u, int v);
void DFS(int adj[MAX][MAX], int v, int s, bool
        visited[MAX]) {
    visited[s] = true;
    printf("%d ", s);
    for (int i=0; i<v; i++) {
        if (adj[s][i] == 1 && !visited[i]){
            DFS(adj, v, i, visited);
        }
    }
}
```

Date
Page

```
int main() {
    int v = 5;
    int adj[MAX][MAX] = {0};
    addEdge(adj, 0, 1);
    addEdge(adj, 1, 3);
    addEdge(adj, 0, 2);
    addEdge(adj, 1, 4);
    addEdge(adj, 3, 4);

    printf("DFS traversal from 0: \n");
    bool visited[MAX] = {false};
    DFS(adj, v, 0, visited);
    printf("BFS traversal from 0: \n");
    BFS(adj, v, 0);
    return 0;
}

void addEdge(int adj[MAX][MAX], int u, int v){
    adj[u][v] = 1; adj[v][u] = 1;
}
```

	0	1	2	3	4
0	0	1	1	0	0
1	1	0	0	1	1
2	0	0	0	0	1
3	0	1	0	0	0
4	0	1	0	0	0



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;
}
```

```
PS C:\Users\Shashank U\Desktop\C> gcc .\Graph.c
PS C:\Users\Shashank U\Desktop\C> .\a.exe
DFS Traversal with initial node 0:0 1 3 4 2
BFS Traversal with initial node 0:0 1 2 3 4
PS C:\Users\Shashank U\Desktop\C>
```