VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Data Structures using C Lab

(23CS3PCDST)

Submitted by

SHASHANK GOWDA L (1BM23CS311)

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

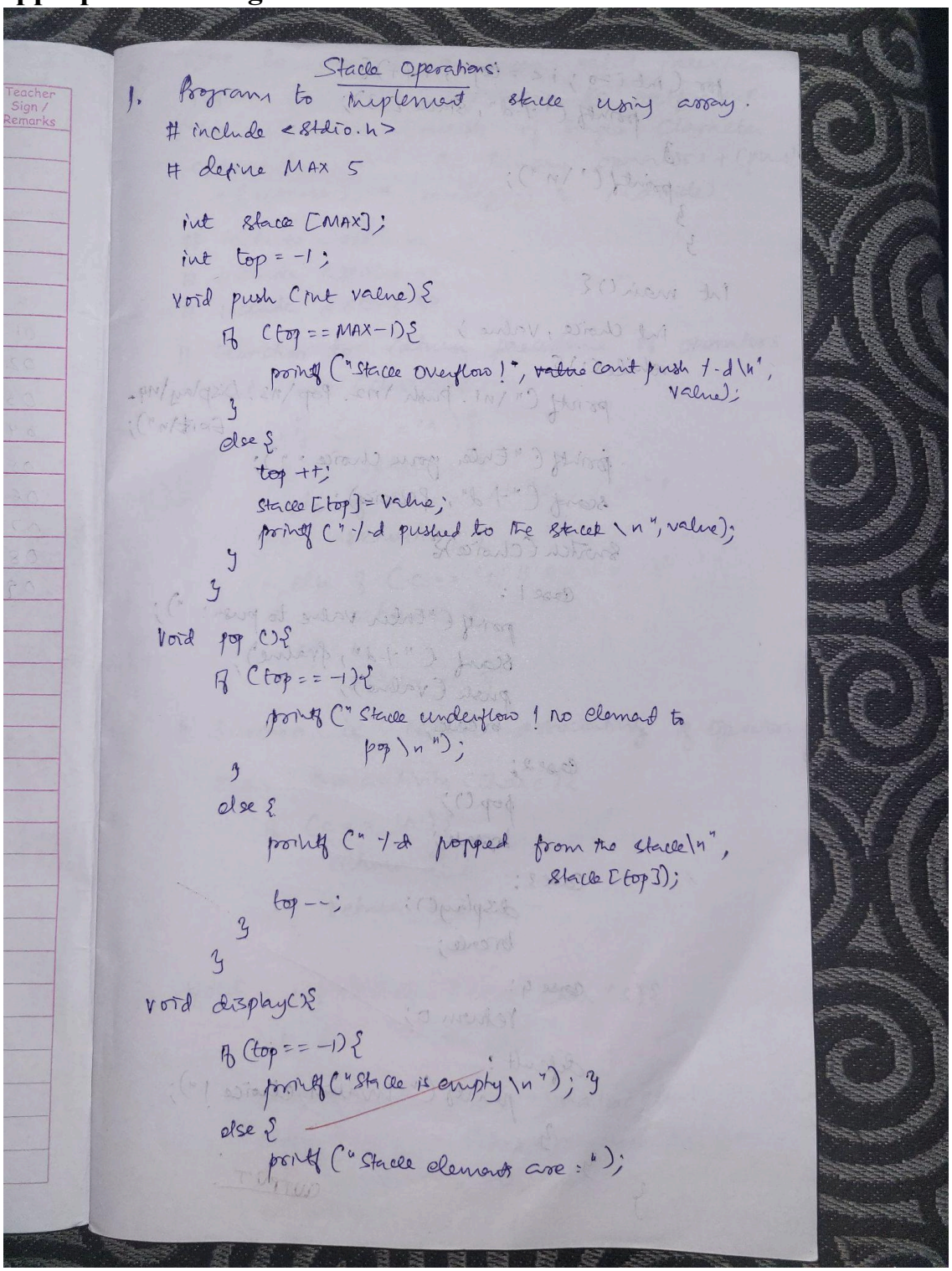
This is to certify that the Lab work entitled "Data Structures using C Lab (23CS3PCDST)"carried out by **SHASHANK GOWDA L(1BM23CS311),** who is bonafide student of **B.M.S.College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering inComputer Science and Engineering** 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 Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

Index

Github Link: https://github.com/ShashankGowdaL69/1BM23CS311_DS_Lab

Question 1: Write a program to simulate the working of a stack using an array with the following: a) Push b) Pop c) Display The program should print appropriate messages for stack overflow and stack underflow.



Stack operations:

1. Program to implement stack using array.

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

int stack [MAX];
int top = -1;
void push (int value){
    if (top == MAX-1){
        printf ("Stack Overflow!", ~~value~~ Cant push %d\n",
                                    value);
    }
    else{
        top ++;
        stack [top] = value;
        printf ("%d pushed to the stack \n", value);
    }
}
void pop (){
    if (top == -1){
        printf (" Stack underflow ! no element to
                pop \n");
    }
    else{
        printf (" %d popped from the stack\n",
                                    stack [top]);
        top --;
    }
}
void display(){
    if (top == -1){
        printf ("Stack is empty \n"); }
    else{
        printf ("Stack elements are : ");
```

```
    for (int i=0; i<=top; i++){
        printf("%d", stack[i]);
    }
    printf("\n");
  }
}

int main(){
    int choice, value;
    while(1){
        printf("\n1. Push \n2. Pop \n3. Display \n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch(choice){
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push(value);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice!\n");
        }
    }
}
```

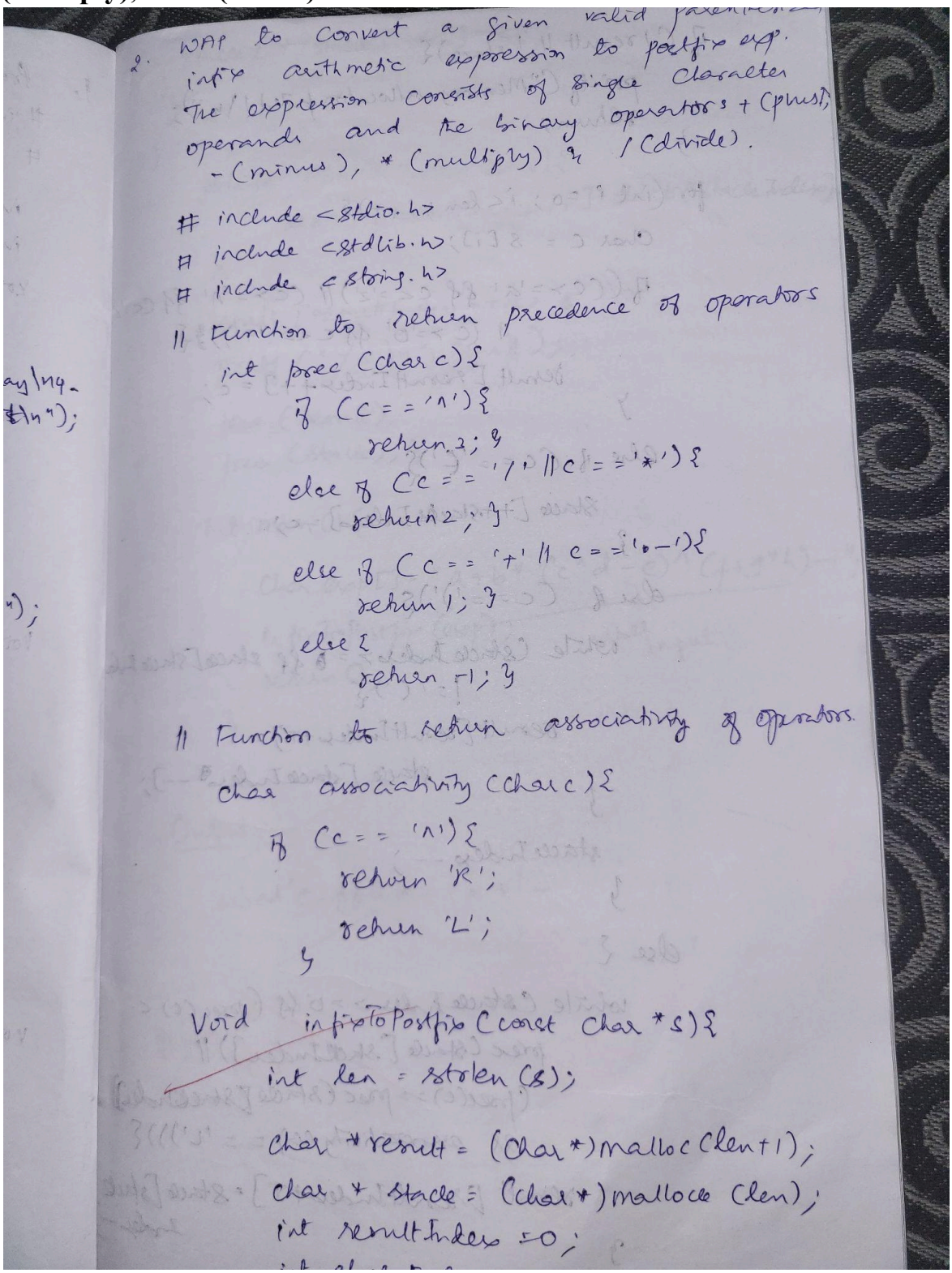
OUTPUT

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

int stack[MAX], top = -1;

void push() {
   int value;
   if (top == MAX - 1) {
      printf("Stack Overflow\n");
   } else {
      printf("Enter value to push: ");
      scanf("%d", &value);
      stack[++top] = value;
      printf("Pushed %d\n", value);
   }
}

void pop() {
   if (top == -1) {
      printf("Stack Underflow\n");
   } else {
      printf("Popped %d\n", stack[top--]);
   }
}

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

int main() {
   int choice;
   while (1) {
      printf("\n1. Push\n2. Pop\n3. Display\n4. Exit\nEnter your choice: ");
      scanf("%d", &choice);
      switch (choice) {
         case 1: push(); break;
         case 2: pop(); break;
         case 3: display(); break;
         case 4: return 0;
```

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

```
cd "/Users/shashank/Documents/dslab/" && gcc 1.c -
shashank@Shashanks-MacBook-Air dslab % cd "/Users/

1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 10
Pushed 10

1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack: 10

1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 4
shashank@Shashanks-MacBook-Air dslab %
```

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



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

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// Function to return precedence of operators
int prec(char c){
    if(c=='^'){
        return 3;}
    else if(c=='/'||c=='*'){
        return 2;}
    else if(c=='+'||c=='-'){
        return 1;}
    else{
        return -1;}
// Function to return associativity of operators.
char associativity(char c){
    if(c=='^'){
        return 'R';
        return 'L';
    }
Void infixToPostfix(const char *s){
    int len = strlen(s);
    char *result = (char*)malloc(len+1);
    char * stack = (char*)malloc(len);
    int resultIndex=0;
```

```
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[stackIndex]) ||
            (prec(c) == prec(stack[stackIndex])
            && associativity(c) == 'L'))) {
            result[resultIndex++] = stack[stack
                Index--];
        }
    }
}
```

```
        stack[++stackIndex] = c;
    }
}

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

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

user input

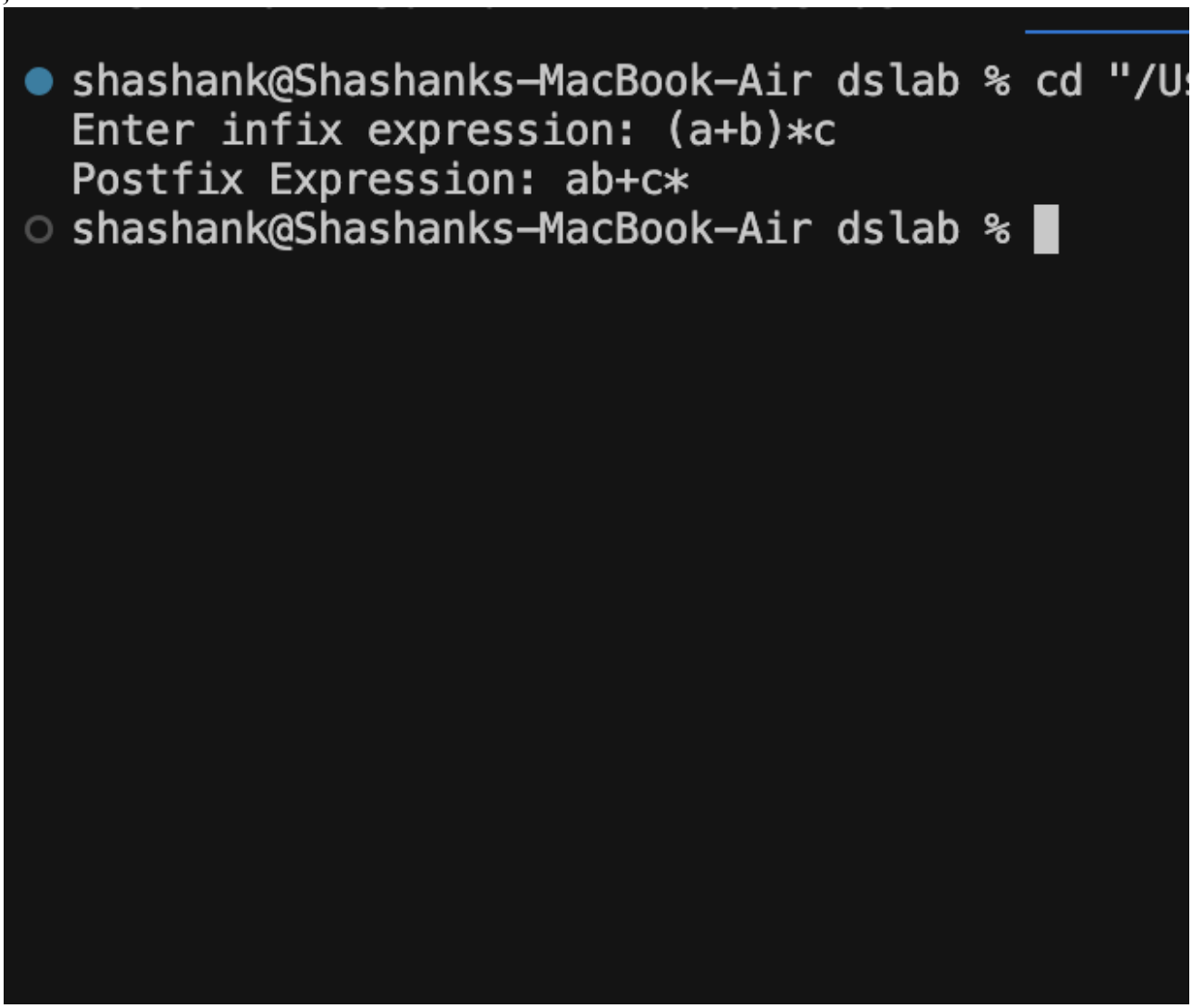
Output:

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

Nanoteja M.
7/10/2024

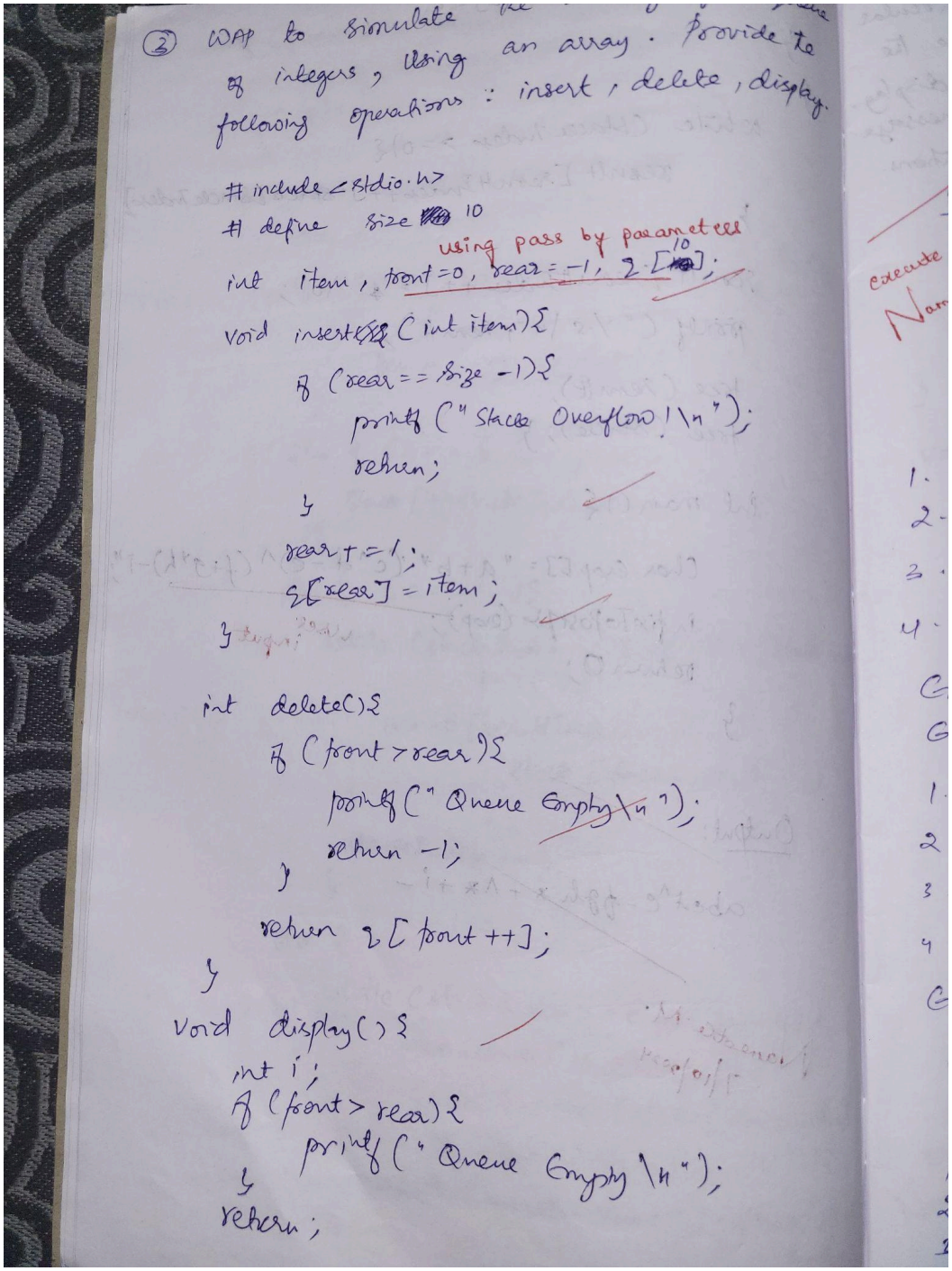
```
#include <stdio.h>
#include <ctype.h>
#include <string.h>

#define MAX 100

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

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

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

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

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

void infixToPostfix(char* exp) {
   char postfix[MAX];
   int k = 0;
   for (int i = 0; exp[i] != '\0'; i++) {
      if (isalnum(exp[i])) {
         postfix[k++] = exp[i];
      } else if (exp[i] == '(') {
         push(exp[i]);
      } else if (exp[i] == ')') {
         while (top != -1 && stack[top] != '(') {
            postfix[k++] = pop();
         }
         pop();
      } else if (isOperator(exp[i])) {
         while (top != -1 && precedence(stack[top]) >= precedence(exp[i])) {
            postfix[k++] = pop();
         }
         push(exp[i]);
      }
   }
```

```
    while (top != -1) {
        postfix[k++] = pop();
    }
    postfix[k] = '\0';
    printf("Postfix Expression: %s\n", postfix);
}

int main() {
    char infix[MAX];
    printf("Enter infix expression: ");
    scanf("%s", infix);
    infixToPostfix(infix);
    return 0;
}
```



Question 3: a)Write a program to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display. The program should print appropriate messages for queue empty and queue overflow conditions.

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



```
(3) WAP to simulate
of integers, using an array. Provide to
following operations : insert, delete, display.

#include <stdio.h>
#define size 10
int item, front=0, rear=-1, q[10];
using pass by parameter
void insert(int item){
    if (rear == size-1){
        printf("Stack Overflow!\n");
        return;
    }
    rear+=1;
    q[rear] = item;
}

int delete(){
    if (front > rear){
        printf("Queue Empty\n");
        return -1;
    }
    return q[front++];
}
void display(){
    int i;
    if (front > rear){
        printf("Queue Empty\n");
    }
    return;
```

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

o/p Seen

Gk
14/10/24

Executed
Namratha. M.
14/10/2024

Output:

Enter your Choice: 4

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

Enter your Choice: 1
Enter Item to insert: 10

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

Enter your Choice: 3

Queue Contains:

10

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

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

```
# include <stdio.h>
# define SIZE 5
int queue[SIZE];   pass by reference
~~front = -1, rear = -1;~~
void insert (int value, int front, int rear){
    if ((front == 0 && rear == SIZE-1) ||
        (rear == (front-1) % (SIZE-1))){
        printf ("Queue Overflow !\n");
        return;
    }
    if (front == -1){
        front = rear = 0;
    }
    else if (rear == SIZE-1 && front != 0){
        rear = 0;
    }
    else {
        ~~rear++;~~ rear = (rear+1) % SIZE;
    }
    queue [rear] = value;
}
Void delete(){
    if (front == -1){
        printf ("Queue Underflow ! \n");
        return;
    }
```

a circular
ovide the
nd display.
e message
onditions.

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

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

execute
Namrata. M.
21/10/2024

Output:

Choices:

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

Enter your choice : 1

Enter a value to insert : 10

Inserted 10

Choices:

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

Enter your choice : 1

Enter a value to insert: 20

Inserted 20

Choices:

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

Enter your Choice : 3

Queue elements : 10 20

Choices:

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

Enter your choice : 2

Deleted 10

choices:

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

Enter your choice : 2

Deleted 20

choices:

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

Enter your choice : 2

Queue Underflow !

choices :

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

Enter your choice : 4

Exiting.

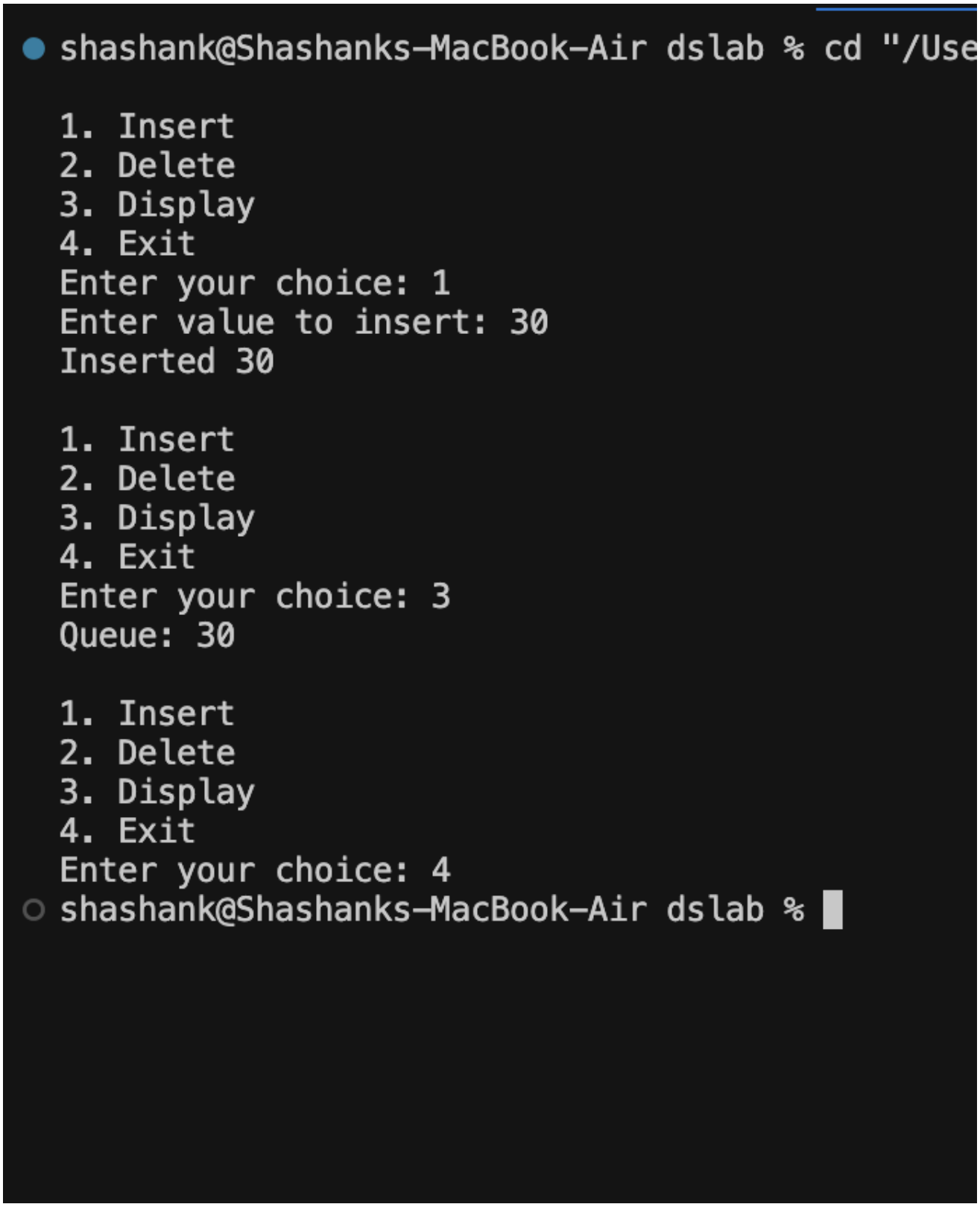
```
#include <stdio.h>
#define MAX 5

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

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

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

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

int main() {
   int choice;
```

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

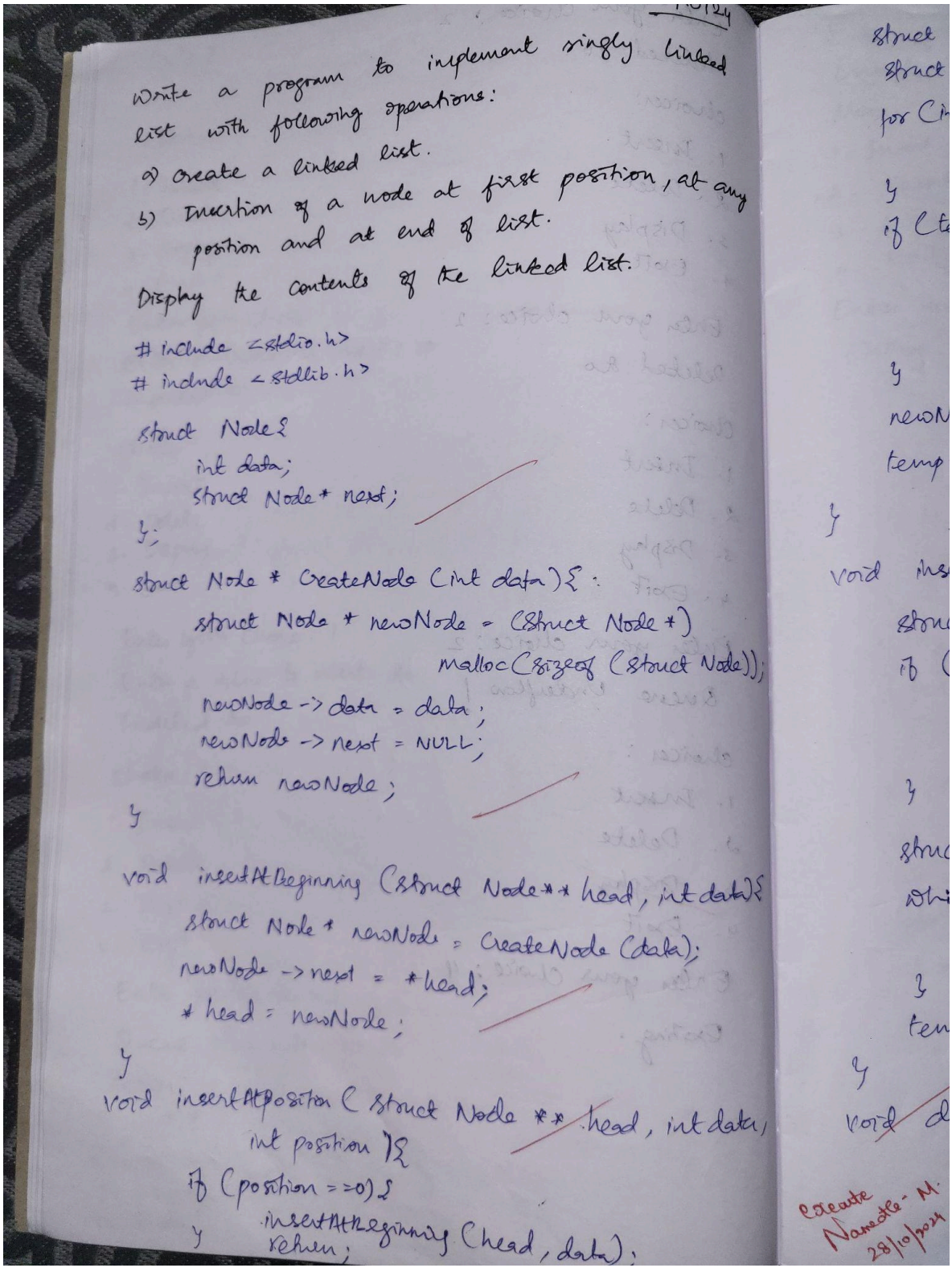
```
#include <stdio.h>
#define MAX 5

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

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

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

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

void display() {
    if (front == -1) {
        printf("Queue is empty\n");
    } else {
        printf("Queue: ");
        int i = front;
        while (i != rear) {
            printf("%d ", queue[i]);
            i = (i + 1) % MAX;
        }
        printf("%d\n", queue[rear]);
    }
}

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

```
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 10
Inserted 10

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

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 4
shashank@Shashanks-MacBook-Air dslab %
```


shashank@Shashanks-MacBook-Air dslab % cd "/Use
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 30
Inserted 30
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue: 30
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 4
shashank@Shashanks-MacBook-Air dslab %

Question 4: Write a program to implement a Singly Linked List with the following operations: a) Create a linked list. b) Insertion of a node at the first position, at any position, and at the end of the list. Display the contents of the linked list.



Write a program to implement singly linked list with following operations:
a) Create a linked list.
b) Insertion of a node at first position, at any position and at end of list.
Display the contents of the linked list.

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

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

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

void insertAtBeginning (struct Node** head, int data){
    struct Node * newNode = CreateNode (data);
    newNode -> next = *head;
    *head = newNode;
}
void insertAtPosition ( struct Node ** head, int data,
        int position ){
    if (position == 0){
        insertAtBeginning (head, data);
        return;
```

Create Namaste - M
28/10/2024

```
    struct Node* newNode = createNode(...);
    struct Node* temp = *head;
    for(int i=0; i<position-1 && temp!=NULL; i++){
        temp = temp->next;
    }
    if(temp==NULL){
        printf("position out of bounds\n");
        free(newNode);
        return;
    }
    newNode->next = temp->next;
    temp->next = 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 displayList(struct Node* head){
    struct Node* temp = head;
    while(temp!=NULL){
```

Create Namaste- M. 2024

printf("NULL\n");

}

Output:

Menu:
1. Insert at Beginning
2. Insert at End
3. Display list
4. Exit
Enter your choice : 1
Enter data to insert at beginning : 12

Menu:
1. Insert at Beginning
2. Insert at End
3. Display list
4. Exit
Enter your choice : 1
Enter data to insert at beginning : 45

Menu:
1. Insert at Beginning
2. Insert at End
3. Display list
4. Exit
Enter your choice : 2
Enter data to insert at end : 89

Menu:
1. Insert at Beginning
2. Insert at End
3. Display list
4. Exit

Enter your choice : 3
Linked List: 45 -> 12 -> 89 -> NULL
Menu:
1. Insert at Beginning
2. Insert at End
3. Display list
4. Exit
Enter your Choice : 4
Exiting.

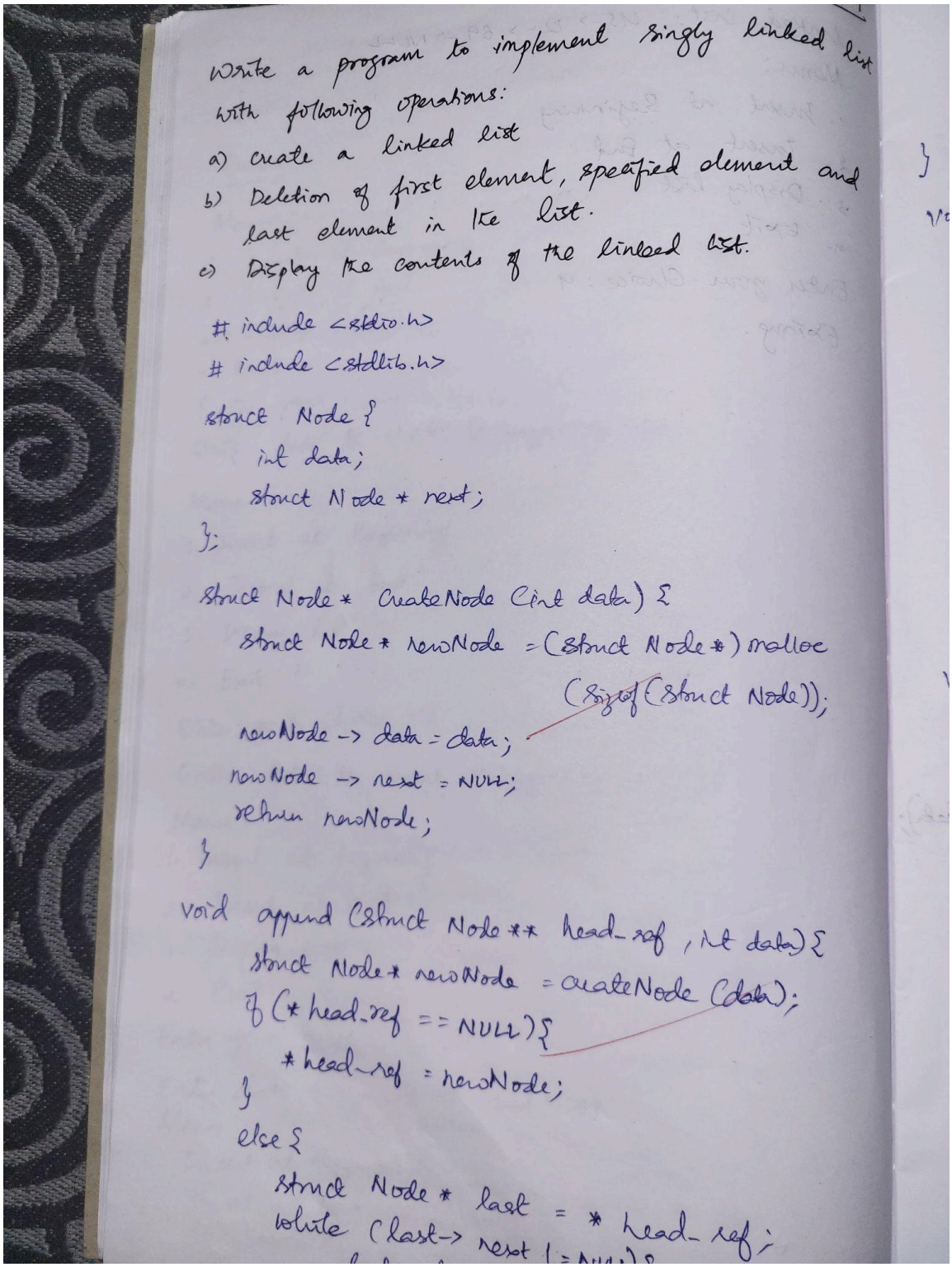
```
#include <stdio.h>
#include <stdlib.h>

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

struct Node* head = NULL;

void create() {
    int n, data;
    printf("Enter number of nodes: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
        printf("Enter data for node %d: ", i + 1);
        scanf("%d", &data);
        newNode->data = data;
        newNode->next = head;
        head = newNode;
    }
}

void insertFirst() {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    printf("Enter data to insert at first position: ");
    scanf("%d", &newNode->data);
    newNode->next = head;
    head = newNode;
}

void insertAtPosition() {
    int pos, data;
    printf("Enter position to insert: ");
    scanf("%d", &pos);
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    printf("Enter data to insert: ");
    scanf("%d", &newNode->data);
    struct Node* temp = head;
```

```
    for (int i = 1; i < pos - 1 && temp != NULL; i++) {
        temp = temp->next;
    }
    if (temp != NULL) {
        newNode->next = temp->next;
        temp->next = newNode;
    } else {
        printf("Position out of range.\n");
        free(newNode);
    }
}

void insertEnd() {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    printf("Enter data to insert at end: ");
    scanf("%d", &newNode->data);
    newNode->next = NULL;
    if (head == NULL) {
        head = newNode;
    } else {
        struct Node* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}

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

```
}

int main() {
   int choice;
   while (1) {
      printf("\n1. Create Linked List\n2. Insert at First\n3. Insert at Position\n4. Insert at
End\n5. Display\n6. Exit\nEnter your choice: ");
      scanf("%d", &choice);
      switch (choice) {
         case 1: create(); break;
         case 2: insertFirst(); break;
         case 3: insertAtPosition(); break;
         case 4: insertEnd(); break;
         case 5: display(); break;
         case 6: return 0;
         default: printf("Invalid choice\n");
      }
   }
}
```

```
shashank@Shashanks-MacBook-Air dslab % cd "/Users/shash

  1. Create Linked List
  2. Insert at First
  3. Insert at Position
  4. Insert at End
  5. Display
  6. Exit
  Enter your choice: 1
  Enter number of nodes: 3
  Enter data for node 1: 10
  Enter data for node 2: 20
  Enter data for node 3: 30

  1. Create Linked List
  2. Insert at First
  3. Insert at Position
  4. Insert at End
  5. Display
  6. Exit
  Enter your choice: 5
  Linked List: 30 20 10

  1. Create Linked List
  2. Insert at First
  3. Insert at Position
  4. Insert at End
  5. Display
  6. Exit
  Enter your choice: 6
shashank@Shashanks-MacBook-Air dslab %
```

Question 5: Write a program to implement a Singly Linked List with the following operations: a) Create a linked list. b) Deletion of the first element, a specified element, and the last element in the list. Display the contents of the linked list.



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.

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

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

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

void append (struct Node ** head_ref , int data) {
    struct Node * newNode = createNode (data);
    if (* head_ref == NULL) {
        * head_ref = newNode;
    }
    else {
        struct Node * last = * head_ref;
        while (last -> next
```

11/11/24

ked list

and

```
    last->next = newNode;
}
printf("Added %d to the list.\n", data);
}
void deleteFirst(struct Node** head_ref) {
    if (*head_ref == NULL) {
        printf("List is empty. Nothing to delete.\n");
        return;
    }
    struct Node* temp = *head_ref;
    *head_ref = (*head_ref)->next;
    printf("Deleted first element: %d\n", temp->data);
    free(temp);
}
void deleteElement(struct Node** head_ref, int key) {
    struct Node* temp = *head_ref, *prev = NULL;
    if (temp != NULL && temp->data == key) {
        *head_ref = temp->next;
        printf("Deleted specified element: %d\n",
               key);
        return;
    }
    while (temp != NULL && temp->data != key) {
        prev = temp;
        temp = temp->next;
    }
```

```
    if (temp == NULL){
        printf("Element %d not found.\n");
        return;
    }
    prev->next = temp->next;
    printf("Deleted element : %d\n", key);
    free(temp);
}

void deleteLast(struct Node** head_ref){
    if(*head_ref == NULL){
        printf("List is empty.\n");
        return;
    }
    struct Node* temp = *head_ref;
    if(temp->next == NULL){
        printf("Deleted last element : %d\n", temp->data);
        free(temp);
        *head_ref = NULL;
        return;
    }
    struct Node* prev = NULL;
    while(temp->next != NULL){
        prev = temp;
        temp = temp->next;
    }
```

```
    printf("Deleted last element: %d\n", temp->data);
    prev->next = NULL;
    free(temp);
}
void displayList(struct Node* node){
    if(node == NULL){
        printf("List is empty.\n");
        return;
    }
    printf("Linked list: ");
    while(node != NULL){
        printf("%d -> ", node->data);
        node = node->next;
    }
    printf("NULL\n");
}
```

->data);

Namrata M.
11/11/2022

6a)

Output:

Menu:

1. Add node
2. Delete first node
3. Delete node by value
4. Delete last node
5. Display list
6. Exit

Enter Choice : 1

Enter value to add : 5

Added 5 to the list

Enter Choice : 1

Enter value to add: 10

Added 10 to the list

Enter choice : 1

Enter value to add: 15

Added 15 to the list

Enter choice : 4

Deleted last element : 15

Enter choice: 5

Linked list: 5 -> 10 -> NULL

Enter choice : 6

Exiting program

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

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

struct Node* head = NULL;

void create() {
   int n, data;
   printf("Enter number of nodes: ");
   scanf("%d", &n);
   for (int i = 0; i < n; i++) {
      struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
      printf("Enter data for node %d: ", i + 1);
      scanf("%d", &data);
      newNode->data = data;
      newNode->next = head;
      head = newNode;
   }
}

void deleteFirst() {
   if (head == NULL) {
      printf("List is empty.\n");
      return;
   }
   struct Node* temp = head;
   head = head->next;
   printf("Deleted first element: %d\n", temp->data);
   free(temp);
}

void deleteSpecified() {
   int data;
   printf("Enter data of element to delete: ");
   scanf("%d", &data);
   if (head == NULL) {
      printf("List is empty.\n");
      return;
   }
   struct Node* temp = head;
   struct Node* prev = NULL;
   while (temp != NULL && temp->data != data) {
      prev = temp;
```

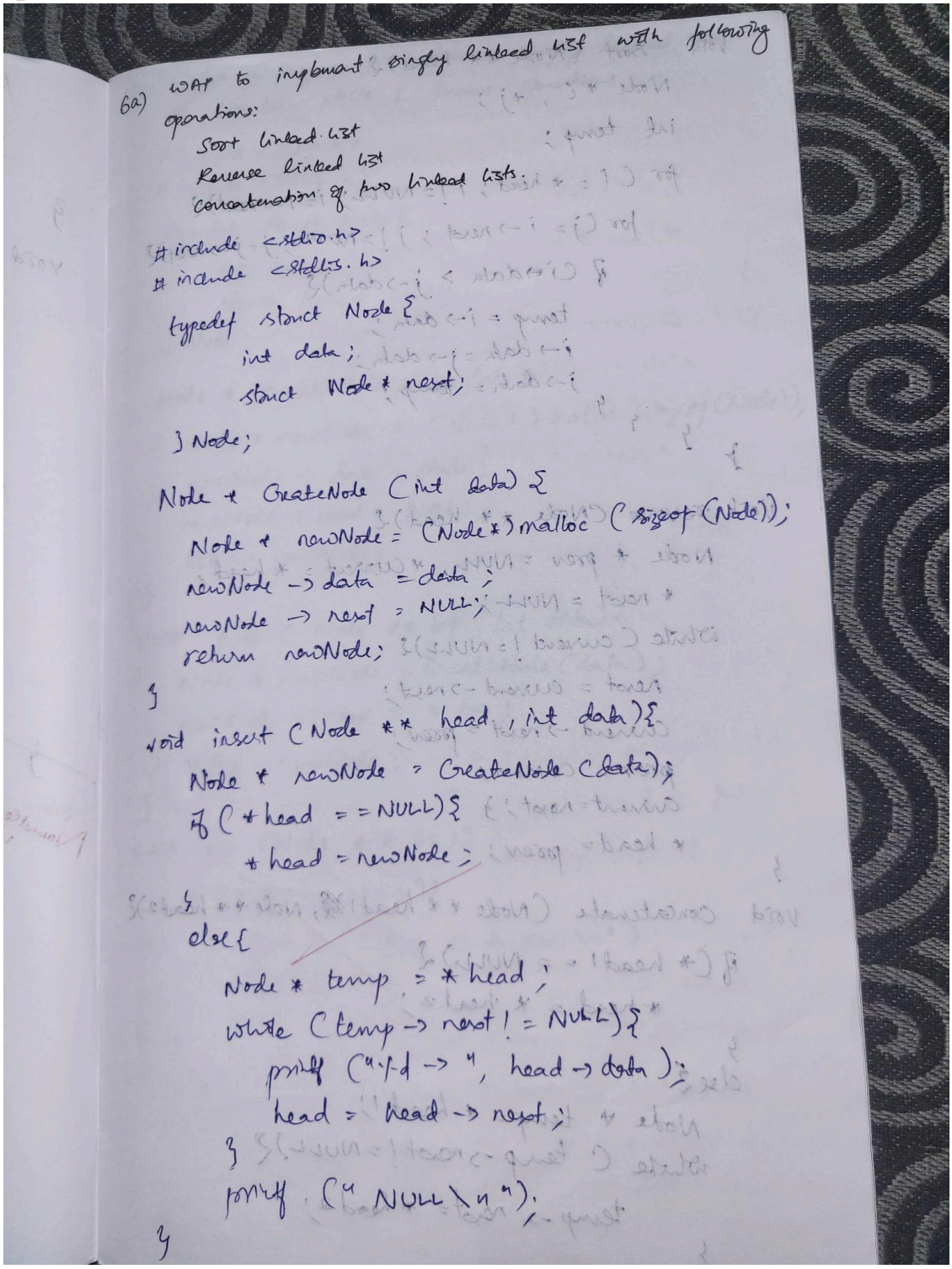
```
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Element not found.\n");
        return;
    }
    if (prev == NULL) {
        head = temp->next;
    } else {
        prev->next = temp->next;
    }
    printf("Deleted element: %d\n", temp->data);
    free(temp);
}

void deleteLast() {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    struct Node* temp = head;
    struct Node* prev = NULL;
    while (temp->next != NULL) {
        prev = temp;
        temp = temp->next;
    }
    if (prev == NULL) {
        head = NULL;
    } else {
        prev->next = NULL;
    }
    printf("Deleted last element: %d\n", temp->data);
    free(temp);
}

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

```
}

int main() {
    int choice;
    while (1) {
        printf("\n1. Create Linked List\n2. Delete First Element\n3. Delete Specified Element\n4. Delete
Last Element\n5. Display\n6. Exit\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1: create(); break;
            case 2: deleteFirst(); break;
            case 3: deleteSpecified(); break;
            case 4: deleteLast(); break;
            case 5: display(); break;
            case 6: return 0;
            default: printf("Invalid choice\n");
        }
    }
}
```

```
shashank@Shashanks-MacBook-Air dslab % cd "/Users/shashar

  1. Create Linked List
  2. Delete First Element
  3. Delete Specified Element
  4. Delete Last Element
  5. Display
  6. Exit
  Enter your choice: 1
  Enter number of nodes: 3
  Enter data for node 1: 30
  Enter data for node 2: 40
  Enter data for node 3: 50

  1. Create Linked List
  2. Delete First Element
  3. Delete Specified Element
  4. Delete Last Element
  5. Display
  6. Exit
  Enter your choice: 4
  Deleted last element: 30

  1. Create Linked List
  2. Delete First Element
  3. Delete Specified Element
  4. Delete Last Element
  5. Display
  6. Exit
  Enter your choice: 6
shashank@Shashanks-MacBook-Air dslab %
```

Question 6: a)Write a program to implement a Singly Linked List with the following operations: a) Sort the linked list. b) Reverse the linked list. c) Concatenate two linked lists.

b) Write a program to Implement Singly Link List to simulate Stack & queue operations.



```
6a) WAP to implement singly linked list with following
operations:
   Sort linked list
   Reverse linked list
   Concatenation of two linked lists.

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

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

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

void insert (Node ** head, int data){
    Node * newNode = CreateNode (data);
    if (*head == NULL){
        *head = newNode;
    }
    else{
        Node * temp = *head;
        while (temp -> next != NULL){
            printf ("%d -> ", head -> data);
            head = head -> next;
        }
        printf ("NULL\n");
    }
```

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

void concatenate (Node ** head1, Node ** head2){
    if (* head1 == NULL){
        *head = * head2;
    }
    else{
        Node * temp = * head1;
        while (temp->next != NULL){
            temp->
```

6b) Write a program to implement singly linked list to simulate stack & queue operations

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{
    int data;
    struct Node* next;
} Node;
Node * createNode (int data){
    Node * newNode = (Node*) malloc (sizeof (Node));
    newNode -> data = data;
    newNode -> next = NULL;
    return newNode;
}
Void push ( Node ** top, int data){
    Node * newNode = CreateNode (data);
    newNode -> next = * top;
    * top = NewNode;
}
void pop (Node * * top){
    if (* top == NULL){
        printf ("Stack underflow\n");
        return -1;
    }
```

```
    Node * temp = * top;
    int data = temp -> data;
    * top = (* top) -> next;
    free (temp);
    return data;
}
void enqueue (Node ** rear, Node * * front, int data){
    Node * newNode = CreateNode (data);
    if (* rear == NULL){
        * rear = * front = newNode;
    }
    else{
        (* rear) -> next = newNode;
        * rear = newNode;
    }
}
int Dequeue (Node * * front){
    if (* front == NULL){
        printf ("Queue Underflow");
        return -1;
    }
    Node * temp = * front;
    int data = temp -> data;
    * front = (* front) -> next;
    free (temp);
    return data;
}
```

Output a

Enter

Enter

Enter

Enter

orginal

orginal

Sorted

~~orgi~~ R

After

Output

1. Pu

2. Po

3. Pe

4. En

5. De

6. Pe

7. E

Enter

Enter

Enter

Enter

Enter

Enter

Enter

Poppe

Output a:

Enter the number of elements for List 1: 2
Enter 2 element for List 1: 1 2
Enter the number of elements for List 2: 3
Enter 3 elements for List 2: 3 4 5

Original List1: 1 -> 2 -> NULL
Original List2: 3 -> 4 -> 5 -> NULL

Sorted List 1: 1 -> 2 -> NULL
Reversed List 2: 5 -> 4 -> 3 -> NULL
After concatenation: 1 -> 2 -> 5 -> 4 -> 3 -> NULL

Output b:

1. Push to Stack
2. Pop from Stack
3. Peek Stack
4. Enqueue to Queue
5. Dequeue from Queue
6. Peek Queue
7. Exit

Enter your choice: 1
Enter data to push: 10
Enter your choice: 1
Enter data to push: 20
Enter your choice: 1
Enter data to push: 30
Enter your choice: 2
Popped from Stack: 30
Enter your choice: 3
Top of Stack: 20
Enter your choice: 4
Enter data to enqueue: 30
Enter your choice: 6
Front of Queue: 30

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

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

struct Node* head = NULL;

void create() {
    int n, data;
    printf("Enter number of nodes: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
        printf("Enter data for node %d: ", i + 1);
        scanf("%d", &data);
        newNode->data = data;
        newNode->next = head;
        head = newNode;
    }
}

void sortList() {
    if (head == NULL) return;
    struct Node* temp1 = head;
    struct Node* temp2 = NULL;
    int tempData;
    while (temp1 != NULL) {
        temp2 = temp1->next;
        while (temp2 != NULL) {
            if (temp1->data > temp2->data) {
                tempData = temp1->data;
                temp1->data = temp2->data;
                temp2->data = tempData;
            }
            temp2 = temp2->next;
        }
        temp1 = temp1->next;
```

```
    }
    printf("List sorted.\n");
}

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

void concatenate(struct Node* secondList) {
    if (head == NULL) {
        head = secondList;
        return;
    }
    struct Node* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = secondList;
    printf("Lists concatenated.\n");
}

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

```
    }
}

int main() {
    int choice;
    while (1) {
        printf("\n1. Create Linked List\n2. Sort List\n3. Reverse List\n4. Concatenate
Lists\n5. Display\n6. Exit\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1: create(); break;
            case 2: sortList(); break;
            case 3: reverseList(); break;
            case 4: {
                struct Node* secondList = NULL;
                create();
                concatenate(secondList); break;
            }
            case 5: display(); break;
            case 6: return 0;
            default: printf("Invalid choice\n");
        }
    }
}
```

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

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

struct Node* top = NULL;
struct Node* front = NULL;
struct Node* rear = NULL;
```

```
void stackPush() {
    int value;
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    printf("Enter value to push: ");
    scanf("%d", &value);
    newNode->data = value;
    newNode->next = top;
    top = newNode;
    printf("Pushed %d onto stack.\n", value);
}

void stackPop() {
    if (top == NULL) {
        printf("Stack is empty.\n");
    } else {
        struct Node* temp = top;
        printf("Popped %d from stack.\n", temp->data);
        top = top->next;
        free(temp);
    }
}


void queueInsert() {
    int value;
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    printf("Enter value to insert in queue: ");
    scanf("%d", &value);
    newNode->data = value;
    newNode->next = NULL;
    if (rear == NULL) {
        front = rear = newNode;
    } else {
        rear->next = newNode;
        rear = newNode;
    }
    printf("Inserted %d in queue.\n", value);
}
```

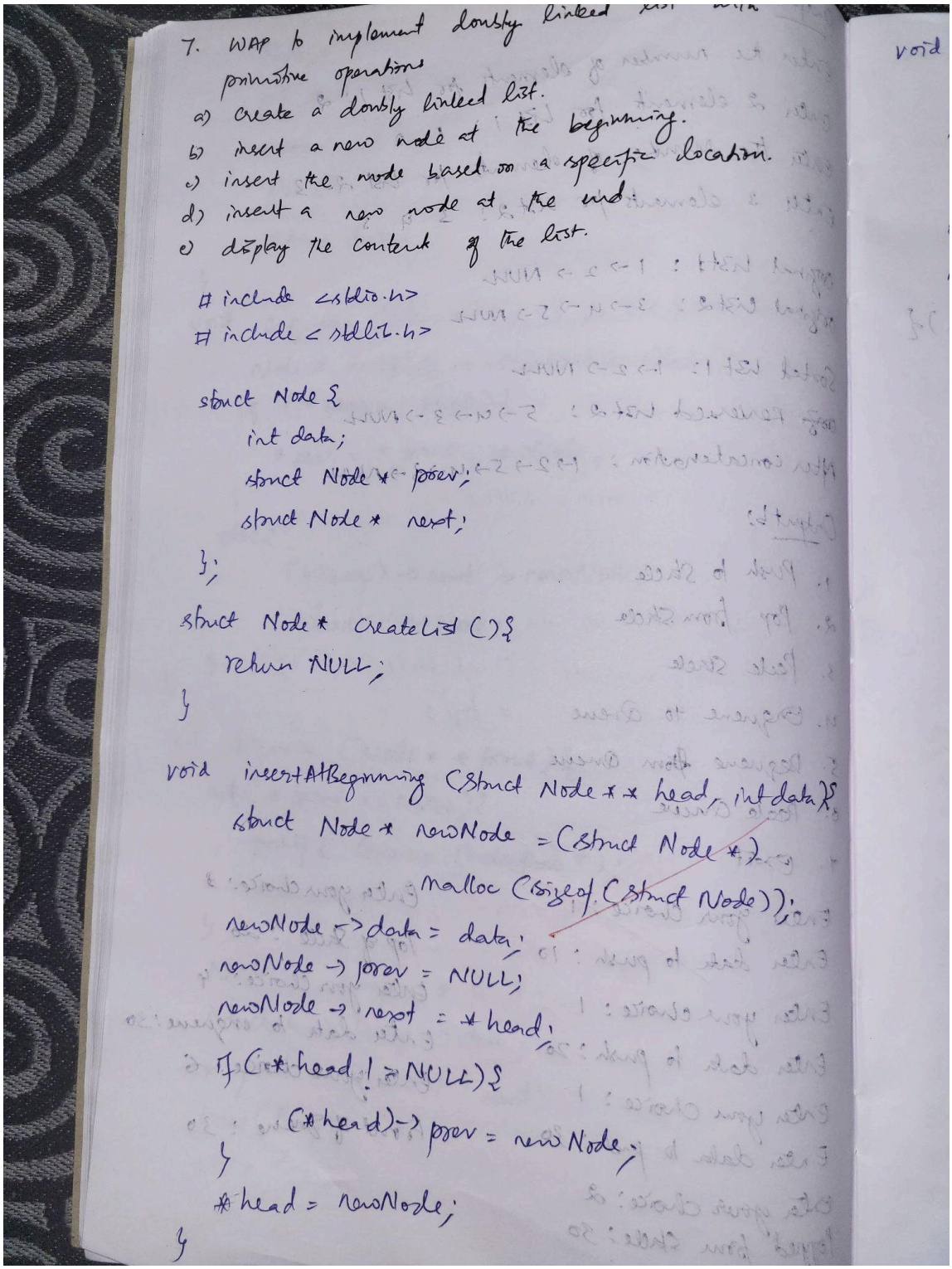
```
void queueDelete() {
   if (front == NULL) {
      printf("Queue is empty.\n");
   } else {
      struct Node* temp = front;
      printf("Deleted %d from queue.\n", temp->data);
      front = front->next;
      if (front == NULL) rear = NULL;
      free(temp);
   }
}

void displayStack() {
   if (top == NULL) {
      printf("Stack is empty.\n");
   } else {
      struct Node* temp = top;
      printf("Stack: ");
      while (temp != NULL) {
         printf("%d ", temp->data);
         temp = temp->next;
      }
      printf("\n");
   }
}

void displayQueue() {
   if (front == NULL) {
      printf("Queue is empty.\n");
   } else {
      struct Node* temp = front;
      printf("Queue: ");
      while (temp != NULL) {
         printf("%d ", temp->data);
         temp = temp->next;
      }
      printf("\n");
   }
}
```

```
int main() {
   int choice;
   while (1) {
      printf("\n1. Stack Push\n2. Stack Pop\n3. Display Stack\n4. Queue Insert\n5.
Queue Delete\n6. Display Queue\n7. Exit\nEnter your choice: ");
      scanf("%d", &choice);
      switch (choice) {
         case 1: stackPush(); break;
         case 2: stackPop(); break;
         case 3: displayStack(); break;
         case 4: queueInsert(); break;
         case 5: queueDelete(); break;
         case 6: displayQueue(); break;
         case 7: return 0;
         default: printf("Invalid choice\n");
      }
   }
}
```

```
shashank@Shashanks-MacBook-Air dslab % cd "/Users/shashank

  1. Create Linked List
  2. Sort List
  3. Reverse List
  4. Concatenate Lists
  5. Display
  6. Exit
  Enter your choice: 1
  Enter number of nodes: 3
  Enter data for node 1: 100
  Enter data for node 2: 200
  Enter data for node 3: 300

  1. Create Linked List
  2. Sort List
  3. Reverse List
  4. Concatenate Lists
  5. Display
  6. Exit
  Enter your choice: 3
  List reversed.

  1. Create Linked List
  2. Sort List
  3. Reverse List
  4. Concatenate Lists
  5. Display
  6. Exit
  Enter your choice: 5
  Linked List: 100 200 300

  1. Create Linked List
  2. Sort List
  3. Reverse List
  4. Concatenate Lists
  5. Display
  6. Exit
  Enter your choice: 6
shashank@Shashanks-MacBook-Air dslab %
```

```
shashank@Shashanks-MacBook-Air dslab % cd "/Users/shash

 1. Stack Push
 2. Stack Pop
 3. Display Stack
 4. Queue Insert
 5. Queue Delete
 6. Display Queue
 7. Exit
 Enter your choice: 1
 Enter value to push: 20
 Pushed 20 onto stack.

 1. Stack Push
 2. Stack Pop
 3. Display Stack
 4. Queue Insert
 5. Queue Delete
 6. Display Queue
 7. Exit
 Enter your choice: 4
 Enter value to insert in queue: 100
 Inserted 100 in queue.

 1. Stack Push
 2. Stack Pop
 3. Display Stack
 4. Queue Insert
 5. Queue Delete
 6. Display Queue
 7. Exit
 Enter your choice: 6
 Queue: 100

 1. Stack Push
 2. Stack Pop
 3. Display Stack
 4. Queue Insert
 5. Queue Delete
 6. Display Queue
 7. Exit
 Enter your choice: 7
shashank@Shashanks-MacBook-Air dslab %
```

Question 7: Write a program to implement a Doubly Linked List with the following operations: a) Create a doubly linked list. b) Insert a new node to the left of the node. c) Delete the node based on a specific value. d) Display the contents of the list.



7. WAP to implement doubly linked list with primitive operations
a) create a doubly linked list.
b) insert a new node at the beginning.
c) insert the node based on a specific location.
d) insert a new node at the end.
e) display the content of the list.

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

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

struct Node * createList (){
    return NULL;
}

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

```
void insertAtPosition(struct Node** head, int data,
    int position){
struct Node* newNode = (struct Node*)
                malloc(sizeof(struct Node));
struct Node* temp = *head;
newNode->data = data;
if(position==0){
    newNode->prev = NULL;
    newNode->next = *head;
    if(*head != NULL){
        (*head)->prev = newNode;
    }
    *head = newNode;
    return;
}
for(int i=0; temp!=NULL && i<position-1; i++){
    temp = temp->next;
}
if(temp==NULL){
    printf("position out of range!\n");
    return;
}
newNode->prev = temp;
newNode->next = temp->next;
```

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

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

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

Output:

1. Create a Doubly Linked list
2. Insert left of Node
3. Delete Node
4. Display
5. Exit

Enter your choice: 1
Enter number of Nodes: 3
Enter data for node 1: 20
Enter data for node 2: 30
Enter data for node 3: 40

Enter your choice: 3
Enter value of node to delete: 30
Deleted node with value 30.
Enter your choice: 5

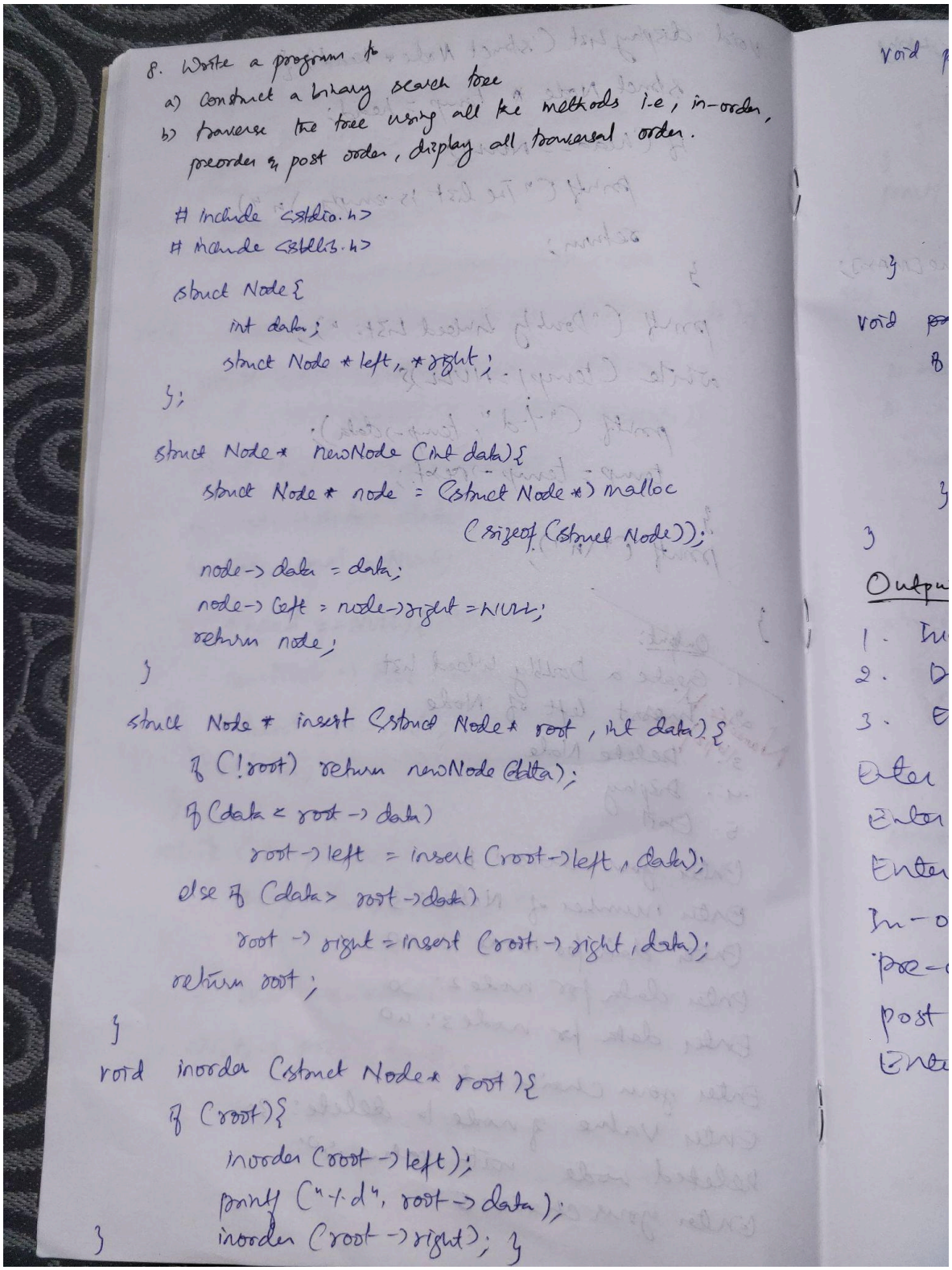
```
#include <stdio.h>
#include <stdlib.h>

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

struct Node* head = NULL;

void create() {
    int n, data;
    printf("Enter number of nodes: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
        printf("Enter data for node %d: ", i + 1);
        scanf("%d", &data);
        newNode->data = data;
        newNode->prev = NULL;
        newNode->next = head;
        if (head != NULL) {
            head->prev = newNode;
        }
        head = newNode;
    }
}

void insertLeftOfNode() {
    int value, newData;
    printf("Enter value of the node to insert left of: ");
    scanf("%d", &value);
    printf("Enter new data to insert: ");
    scanf("%d", &newData);

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

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

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

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

    printf("Inserted %d to the left of %d.\n", newData, value);
}

void deleteNode() {
    int value;
    printf("Enter value of node to delete: ");
    scanf("%d", &value);

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

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

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

```
    }

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

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

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

int main() {
    int choice;
    while (1) {
        printf("\n1. Create Doubly Linked List\n2. Insert Left of Node\n3. Delete
Node\n4. Display\n5. Exit\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1: create(); break;
            case 2: insertLeftOfNode(); break;
            case 3: deleteNode(); break;
            case 4: display(); break;
            case 5: return 0;
            default: printf("Invalid choice\n");
        }
    }
}
```

```
shashank@Shashanks-MacBook-Air dslab % cd "/Users/s

  1. Create Doubly Linked List
  2. Insert Left of Node
  3. Delete Node
  4. Display
  5. Exit
  Enter your choice: 1
  Enter number of nodes: 3
  Enter data for node 1: 20
  Enter data for node 2: 30
  Enter data for node 3: 40

  1. Create Doubly Linked List
  2. Insert Left of Node
  3. Delete Node
  4. Display
  5. Exit
  Enter your choice: 3
  Enter value of node to delete: 30
  Deleted node with value 30.

  1. Create Doubly Linked List
  2. Insert Left of Node
  3. Delete Node
  4. Display
  5. Exit
  Enter your choice: 5
shashank@Shashanks-MacBook-Air dslab %
```

Question 8: Write a program to construct a binary search tree. Traverse the tree using all methods (in-order, pre-order, and post-order) and display the elements of the tree.



8. Write a program to
a) Construct a binary search tree
b) Traverse the tree using all the methods i.e, in-order, preorder & post order, display all traversal order.

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

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

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

struct Node* insert(struct Node* root, int data){
    if(!root) return newNode(data);
    if(data < root->data)
        root->left = insert(root->left, data);
    else if(data > root->data)
        root->right = insert(root->right, data);
    return root;
}

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

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

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

Output:

1. Insert Node
2. Display Tree
3. Exit

Enter your choice: 1
Enter value to insert : 100
Enter your choice: 2
In-order : 100
pre-order : 100
post order : 100
Enter your choice: 3

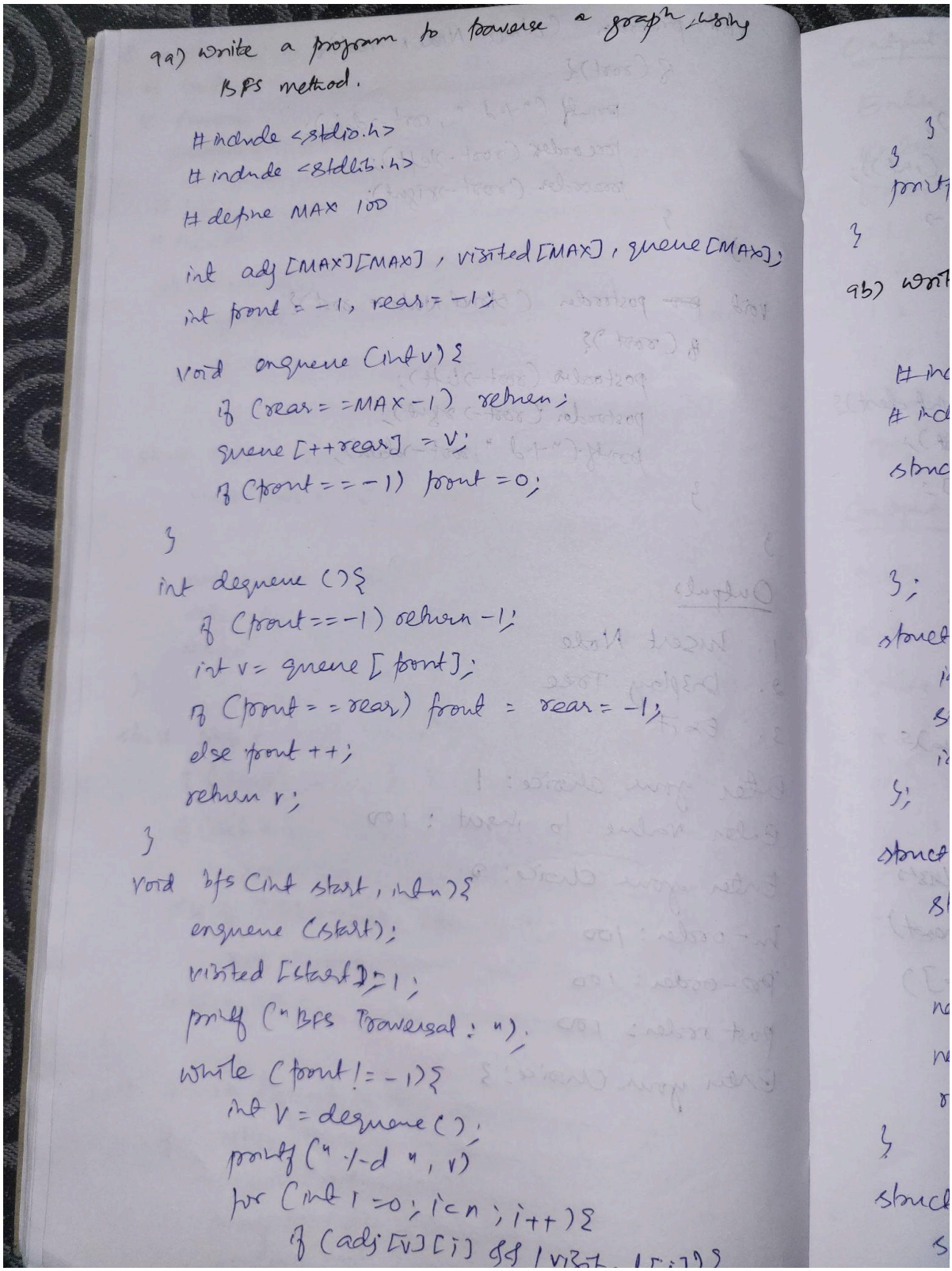
```
#include <stdio.h>
#include <stdlib.h>

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

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

struct Node* insert(struct Node* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }
    if (data < root->data) {
        root->left = insert(root->left, data);
    } else {
        root->right = insert(root->right, data);
    }
    return root;
}

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

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

```
        preorder(root->right);
    }
}

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

void display(struct Node* root) {
    if (root != NULL) {
        printf("In-order: ");
        inorder(root);
        printf("\nPre-order: ");
        preorder(root);
        printf("\nPost-order: ");
        postorder(root);
        printf("\n");
    } else {
        printf("Tree is empty.\n");
    }
}

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

    while (1) {
        printf("\n1. Insert Node\n2. Display Tree\n3. Exit\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                root = insert(root, value);
                break;
            case 2:
```

```
            display(root);
            break;
        case 3:
            return 0;
        default:
            printf("Invalid choice\n");
    }
  }
}
```

```
shashank@Shashanks-MacBook-Air dslab % cd "/Users

1. Insert Node
2. Display Tree
3. Exit
Enter your choice: 1
Enter value to insert: 100

1. Insert Node
2. Display Tree
3. Exit
Enter your choice: 2
In-order: 100
Pre-order: 100
Post-order: 100

1. Insert Node
2. Display Tree
3. Exit
Enter your choice: 3
shashank@Shashanks-MacBook-Air dslab %
```

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



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

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

int adj[MAX][MAX], visited[MAX], queue[MAX];
int front = -1, rear = -1;

void enqueue(int v){
    if(rear == MAX-1) return;
    queue[++rear] = v;
    if(front == -1) front = 0;
}
int dequeue(){
    if(front == -1) return -1;
    int v = queue[front];
    if(front == rear) front = rear = -1;
    else front++;
    return v;
}
void bfs(int start, int n){
    enqueue(start);
    visited[start] = 1;
    printf("BFS Traversal: ");
    while(front != -1){
        int v = dequeue();
        printf("%d ", v);
        for(int i=0; i<n; i++){
            if(adj[v][i] && !visited[i]){
```

```
            enqueue(i);
            visited[i]=1;
        }
    }
}
printf("\n");
}
```

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

```
#include <stdio.h>
#include <stdlib.h>
struct Node{
    int vertex;
    struct Node* next;
};
struct Graph{
    int numVertices;
    struct Node** adjLists;
    int* visited;
};
struct Node* CreateNode(int v){
    struct Node* newNode = malloc
        malloc(sizeof(struct Node));
    newNode->vertex = v;
    newNode->next = NULL;
    return newNode;
}
struct Graph* CreateGraph(int vertices){
    struct Graph* graph =
```

```
graph->numVertices = vertices;
graph->adjLists = malloc(vertices *
                  sizeof(struct Node *));
graph->visited = calloc(vertices, sizeof(int));
for(int i=0; i<vertices; i++) graph->
  adjLists[i] = NULL;
return graph;
}
void addEdge(struct Graph* graph, int src, int dest){
  struct Node* newNode = CreateNode(dest);
  newNode->next = graph->adjLists[src];
  graph->adjLists[src] = newNode;
  newNode = CreateNode(src);
  newNode->next = graph->adjLists[dest];
  graph->adjLists[dest] = newNode;
}
void DFS(struct Graph* graph, int vertex){
  graph->visited[vertex] = 1;
  printf("%d ", vertex);
  for(struct Node* temp = graph->adjLists
    [vertex]; temp; temp = temp->next)
  if(!graph->visited[temp->vertex])
  DFS(graph, temp->vertex);
}
```

Output

```
Enter
Enter
Enter
0 1
0 2
1 3
1 4
Ente
BFS
Output
Enter
Enter
Enter
0 1
0 2
1
1
The
```

Output: a)

```
Enter number of vertices : 5
Enter number of edges : 4
Enter edges (u v):
0 1
0 2
1 3
1 4
Enter starting vertex for BFS : 0
BFS traversal : 0 1 2 3 4
```

Output : b)

```
Enter number of vertices : 5
Enter number of edges : 4
Enter edges (u v):
0 1
0 2
1 3
1 4
The graph is connected.
```

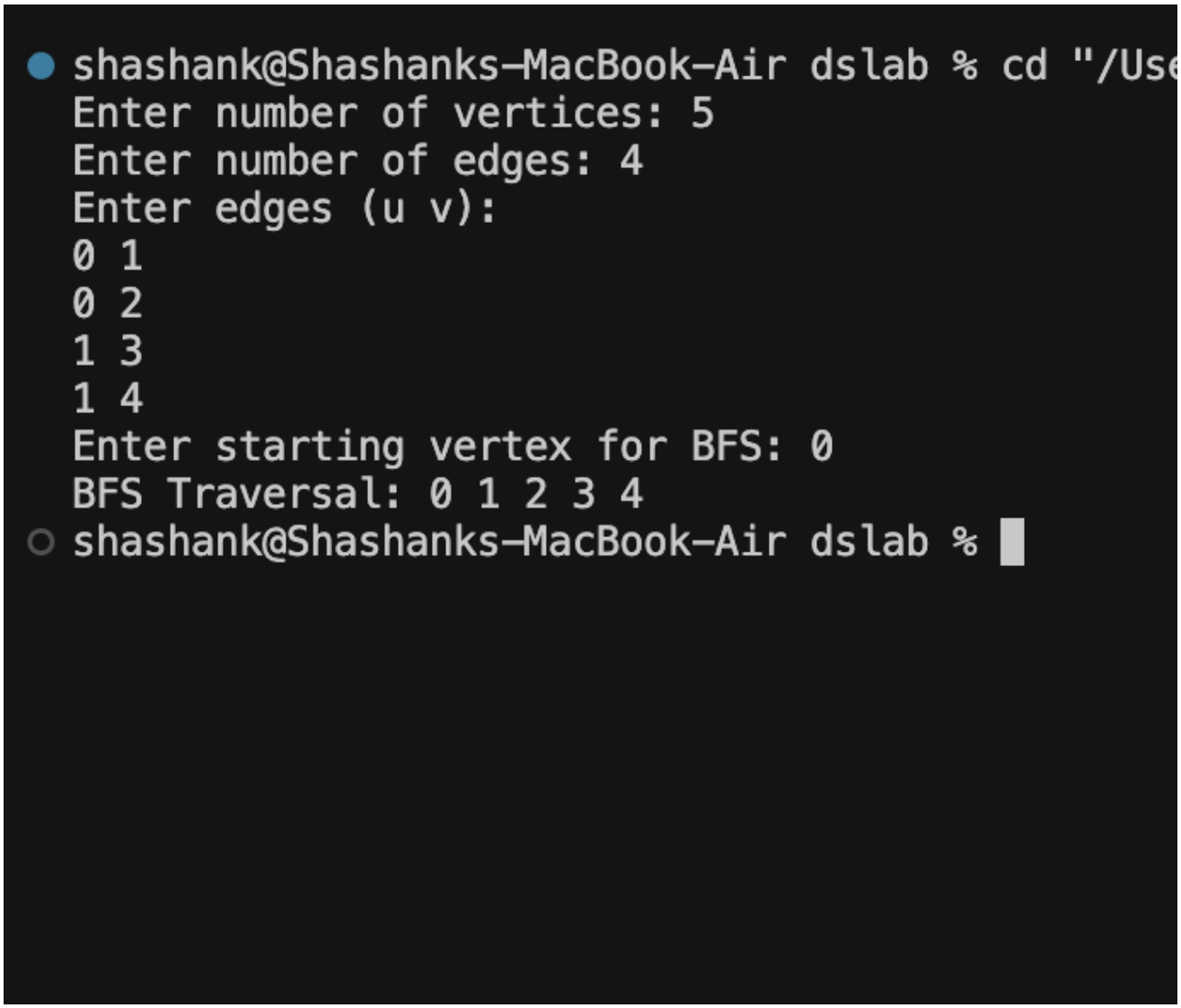
```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define MAX_VERTICES 10

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

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

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

void enqueue(struct Queue* q, int value) {
    if (q->rear == MAX_VERTICES - 1) {
        printf("Queue is full\n");
        return;
    }
    if (q->front == -1) {
        q->front = 0;
    }
    q->rear++;
    q->items[q->rear] = value;
}

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

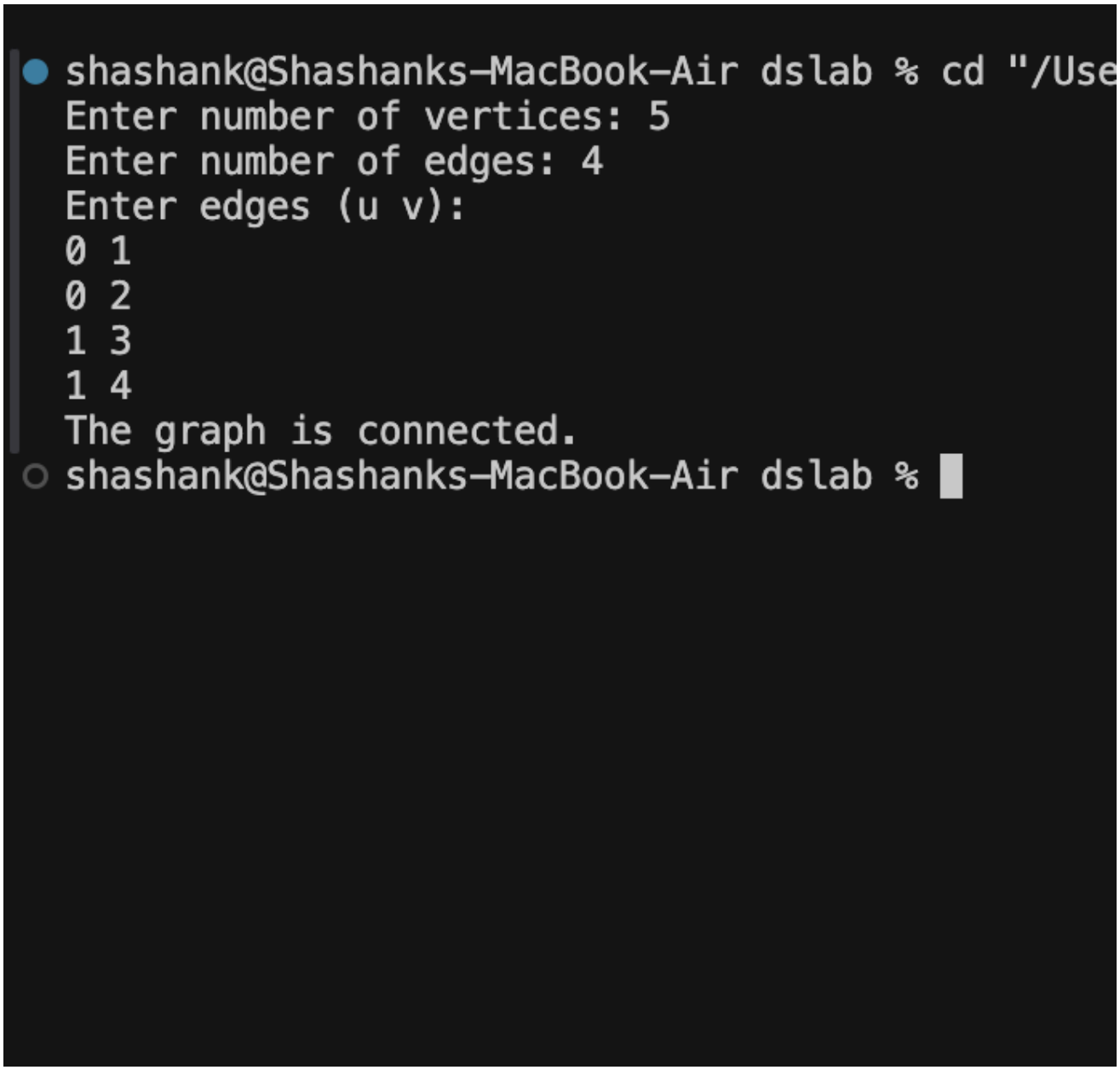
```
    if (q->front > q->rear) {
        q->front = q->rear = -1;
    }
    return value;
}

void bfs(int graph[MAX_VERTICES][MAX_VERTICES], int start, int vertices) {
    bool visited[MAX_VERTICES] = {false};
    struct Queue q;
    initializeQueue(&q);
    enqueue(&q, start);
    visited[start] = true;

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

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

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

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

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

    printf("Enter edges (u v): \n");
    for (int i = 0; i < edges; i++) {
        scanf("%d %d", &u, &v);
        graph[u][v] = 1;
```

```
        graph[v][u] = 1;
    }

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

    printf("BFS Traversal: ");
    bfs(graph, start, vertices);
    printf("\n");

    return 0;
}
```

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

#define MAX_VERTICES 10

void dfs(int graph[MAX_VERTICES][MAX_VERTICES], int vertex, bool visited[],
int vertices) {
    visited[vertex] = true;
    for (int i = 0; i < vertices; i++) {
        if (graph[vertex][i] == 1 && !visited[i]) {
            dfs(graph, i, visited, vertices);
        }
    }
}

bool isConnected(int graph[MAX_VERTICES][MAX_VERTICES], int vertices) {
    bool visited[MAX_VERTICES] = {false};
    dfs(graph, 0, visited, vertices);

    for (int i = 0; i < vertices; i++) {
```

```
        if (!visited[i]) {
            return false;
        }
    }
    return true;
}

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

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

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

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

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

    return 0;
}
```


shashank@Shashanks-MacBook-Air dslab % cd "/Use
Enter number of vertices: 5
Enter number of edges: 4
Enter edges (u v):
0 1
0 2
1 3
1 4
Enter starting vertex for BFS: 0
BFS Traversal: 0 1 2 3 4
shashank@Shashanks-MacBook-Air dslab %


shashank@Shashanks-MacBook-Air dslab % cd "/Use
Enter number of vertices: 5
Enter number of edges: 4
Enter edges (u v):
0 1
0 2
1 3
1 4
The graph is connected.
shashank@Shashanks-MacBook-Air dslab %

Question 10: Given a file of N employee records with a set K of keys (4-digit) that uniquely determine the records in file F, assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L be integers. Design and develop a program in C that uses Hash function H: K -> L as H(K) = K mod m (remainder method) and implement hashing technique to map a given key K to the address space L. Resolve collisions (if any) using linear probing.

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

#define MAX_EMPLOYEES 100


#define M 10
struct Employee {
   int key;
   int data;
};

struct Employee hashTable[M];

void initializeHashTable() {
   for (int i = 0; i < M; i++) {
      hashTable[i].key = -1;
      hashTable[i].data = -1;
   }
}

int hashFunction(int key) {
   return key % M;
}

void insert(int key, int data) {
   int index = hashFunction(key);
   int originalIndex = index;

   while (hashTable[index].key != -1) {
      if (hashTable[index].key == key) {
         printf("Employee with key %d already exists.\n", key);
```

```
            return;
        }
        index = (index + 1) % M;
        if (index == originalIndex) {
            printf("Hash table is full.\n");
           return;
        }
    }

    hashTable[index].key = key;
    hashTable[index].data = data;
    printf("Employee with key %d inserted at address %d.\n", key, index);
}

int search(int key) {
    int index = hashFunction(key);
    int originalIndex = index;

    while (hashTable[index].key != -1) {
        if (hashTable[index].key == key) {
            return hashTable[index].data;
        }
        index = (index + 1) % M;
        if (index == originalIndex) {
            break;
        }
    }

    return -1;
}

void displayHashTable() {
    printf("Hash Table:\n");
    for (int i = 0; i < M; i++) {
        if (hashTable[i].key != -1) {
            printf("Address %d -> Key: %d, Data: %d\n", i, hashTable[i].key,
hashTable[i].data);
        } else {
            printf("Address %d -> Empty\n", i);
        }
```

```
    }
}

int main() {
    int choice, key, data;

    initializeHashTable();

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

        switch (choice) {
            case 1:
              printf("Enter Employee Key (4-digit): ");
                scanf("%d", &key);
                printf("Enter Employee Data: ");
                scanf("%d", &data);
                insert(key, data);
                break;
            case 2:
                printf("Enter Employee Key to search (4-digit): ");
                scanf("%d", &key);
                data = search(key);
                if (data != -1) {
                    printf("Employee with key %d found, Data: %d\n", key, data);
                } else {
                    printf("Employee with key %d not found.\n", key);
                }
                break;
            case 3:
                displayHashTable();
                break;
            case 4:
                exit(0);
            default:
```

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

    return 0;
}
```

```
shashank@Shashanks-MacBook-Air dslab % cd "/Users/shashank/Do

Menu:
1. Insert Employee
2. Search Employee
3. Display Hash Table
4. Exit
Enter your choice: 1
Enter Employee Key (4-digit): 1234
Enter Employee Data: 1001
Employee with key 1234 inserted at address 4.

Menu:
1. Insert Employee
2. Search Employee
3. Display Hash Table
4. Exit
Enter your choice: 1
Enter Employee Key (4-digit): 5678
Enter Employee Data: 2002
Employee with key 5678 inserted at address 8.

Menu:
1. Insert Employee
2. Search Employee
3. Display Hash Table
4. Exit
Enter your choice: 3
Hash Table:
Address 0 -> Empty
Address 1 -> Empty
Address 2 -> Empty
Address 3 -> Empty
Address 4 -> Key: 1234, Data: 1001
Address 5 -> Empty
Address 6 -> Empty
Address 7 -> Empty
Address 8 -> Key: 5678, Data: 2002
Address 9 -> Empty

Menu:
1. Insert Employee
2. Search Employee
3. Display Hash Table
4. Exit
Enter your choice: 4
shashank@Shashanks-MacBook-Air dslab %
```