VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

DATA STRUCTURES

Submitted by

AMSHU G M (1BM21CS019)

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
Oct 2022-Feb 2023

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**" carried out by **AMSHU G M(1BM21CS019),** who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2022-23. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab **- (22CS3PCDST)**work prescribed for the said degree.

Lohit J
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

`

	b) Insertion of a node at first position, at any position and at end of list. c) Display the contents of the linked list.	
6	Part 2: 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.	29-35
7	WAP Implement Single Link List with following operations: a) Sort the linked list. b) Reverse the linked list. c) Concatenation of two linked lists	36-44
8	Write a program to implement Stacks and Queues using a linked list.	45-52
9	WAP Implement doubly link list with primitive operations: a) Create a doubly linked list. b) Insert a new node to the left of the node. c) Delete the node based on a specific value d) Display the contents of the list	53-60
10	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 postorder c) To display the elements in the tree.	61-64

Course Outcome

CO1	Apply the concept of linear and nonlinear data structures.
CO2	Analyze data structure operations for a given problem.
CO3	Design and develop solutions using the operations of linear and nonlinear data structure for a given specification.
CO4	Conduct practical experiments for demonstrating the operations of different data structures.

LAB PROGRAM 1:

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

following:

a) Push

b) Pop

c) Display

The program should print appropriate messages for stack overflow, stack

Underflow

```
//code
#include <stdio.h>
#include <stdlib.h>
#define SIZE 3

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

int isFull (){
   if (top==SIZE-1)
      return 1;
   else
      return 0;
}

int isEmpty (){
   if (top<0)
      return 1;
   else
      return 0;
```

```
}
void push (int x){

    if (isFull()){
        printf ("Stack is full!\n");
    }
    else{
            stack [++top]=x;
    }
}
void pop (){

    if (isEmpty()){
        printf ("Stack is empty\n");
    }
    else{
        stack [top--]=0;
    }
}
void print (){

    printf ("Elements are:\n");
    for (int i=0; i<=top; i++){
        printf ("%d\t",stack[i]);
    }
}
void input (int *a){

    printf ("Enter the elements:\n");
    scanf ("%d",&a);
    push (a);
```

```
}
int main()
{
   int a;
   int user_input;

   while (1){
      printf ("\n1:Push\n2:Pop\n3.Display\n4.Quit\n");
      scanf ("%d",&user_input);

      switch (user_input){

         case 1: input (&a);
         break;

         case 2: pop ();
         break;

         case 3: print ();
         break;

         case 4: exit (0);
         break;
       }
   }

   return 0;
}
```

```
1:Push
2:Pop
3.Display
4.Quit
1
Enter the elements:
1

1:Push
2:Pop
4.Quit
3
Elements are:
1       2       3
1:Push
2:Pop
3.Display
4.Quit
2

1:Push
2:Pop
3.Display
4.Quit
2

1:Push
2:Pop
3.Display
4.Quit
3
Elements are:
1
```

```
Elements are:
1
1:Push
2:Pop
3.Display
4.Quit
2

1:Push
2:Pop
3.Display
4.Quit
2
Stack is empty

1:Push
2:Pop
3.Display
4.Quit
4
```

LAB-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)

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

char stack[MAX];
char infix[MAX], postfix[MAX];
int top = -1;

void push(char);
char pop();
int isEmpty();
void inToPost();
void print();
int precedence(char);

int main()
{
   printf("Enter the infix expression: ");
   gets(infix);
   inToPost();
   print();
```

```
    return 0;
}

void inToPost()
{
    int i, j = 0;
    char symbol, next;
    for (i = 0; i < strlen(infix); i++)
    {
        symbol = infix[i];
        switch (symbol)
        {
        case '(':
            push(symbol);
            break;
        case ')':
            while ((next = pop()) != '(')
                postfix[j++] = next;
            break;
        case '+':
        case '-':
        case '*':
        case '/':
        case '^':
            while (!isEmpty() && precedence(stack[top]) >= precedence(symbol))
                postfix[j++] = pop();
            push(symbol);
            break;
        default:
            postfix[j++] = symbol;
```

```
        }
    }

    while (!isEmpty())
        postfix[j++] = pop();
    postfix[j] = '\0';
}

int precedence(char symbol)
{
    switch (symbol)
    {
    // Higher value means higher precedence
    case '^':
        return 3;
    case '/':
    case '*':
        return 2;
    case '+':
    case '-':
        return 1;
    default:
        return 0;
    }
}

void print()
{
    int i = 0;
    printf("The equivalent postfix expression is: ");
```

```
    while (postfix[i])
    {
        printf("%c", postfix[i++]);
    }
    printf("\n");
}

void push(char c)
{
    if (top == MAX - 1)
    {
        printf("Stack Overflow\n");
        return;
    }
    top++;
    stack[top] = c;
}

char pop()
{
    char c;
    if (top == -1)
    {
        printf("Stack Underflow\n");
        exit(1);
    }
    c = stack[top];
    top = top - 1;
    return c;
}
```

```
int isEmpty()
{
   if (top == -1)
      return 1;
   else
      return 0;
}
```

```
C:\Users\amshu\OneDrive\Desktop\FOLDER\codes>gcc labds.c

C:\Users\amshu\OneDrive\Desktop\FOLDER\codes>labds.exe
Enter the infix expression: 5+4-3/2*5
The equivalent postfix expression is: 54+32/5*-

C:\Users\amshu\OneDrive\Desktop\FOLDER\codes>labds.exe
Enter the infix expression: 5-6+8/2*4
The equivalent postfix expression is: 56-82/4*+
```

LAB-PROGRAM-3

WAP to simulate the working of a queue of integers using an array.

Provide the following operations

a) Insert

b) Delete

c) Display

The program should print appropriate messages for queue empty and

queue overflow conditions

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

#define SIZE 5

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

int isFull (){
   if (rear==SIZE-1)
      return 1;
   else
      return 0;
}
```

```
int isEmpty (){
   if (front==rear)
      return 1;
   else
      return 0;
}

void enqueue (int *x){
   if (isFull())
      printf ("Queue overflow!\n");
   else if (isEmpty()){
      front=0;
      queue[++rear]=*x;
   }
   else
      queue[++rear]=*x;
}

void dequeue (){

   if (isEmpty())
      printf ("Queue underflow!\n");
   else if (front==rear){
      front=-1;
      rear=-1;
   }
   else
      queue[front++]=0;
}
```

```
void display (){
   if (isEmpty())
      printf ("Queue empty!\n");
   else{
      for (int i=front;i<=rear;i++)
         printf ("%d\t", queue[i]);
   }
}

int main()
{
   int choice, input;

   while (1){
      printf ("\n1:Enqueue\n2:Dequeue\n3:Display\n4:Quit\n");
      scanf ("%d",&choice);

      switch (choice){

         case 1:printf ("Enter element to be added:\n");
         scanf ("%d", &input);
         enqueue (&input);
         break;

         case 2:dequeue();
         break;

         case 3:display();
         break;
```

```
        case 4:exit (0);
        break;

        default:printf ("Enter a valid choice!\n");
        break;
      }
   }

   return 0;
}
```

```
1:Enqueue
2:Dequeue
3:Display
4:Quit
1
Enter element to be added:
1

1:Enqueue
2:Dequeue
3:Display
4:Quit
2
Queue underflow!

1:Enqueue
2:Dequeue
3:Display
4:Quit
3
Queue empty!
2:Dequeue
3:Display
4:Quit
1
Enter element to be added:
2
3
Queue empty!
2:Dequeue
3:Display
4:Quit
1
Enter element to be added:
2

1:Enqueue
2:Dequeue
3:Display
4:Quit
2

1:Enqueue
2:Dequeue
3:Display
4:Quit
3
1       2
1:Enqueue
2:Dequeue
3:Display
4:Quit
4
```

LAB-PROGRAM_4

WAP to simulate the working of a circular queue of integers using an

array. Provide the following operations.

a) Insert

b) Delete

c) Display

The program should print appropriate messages for queue empty and

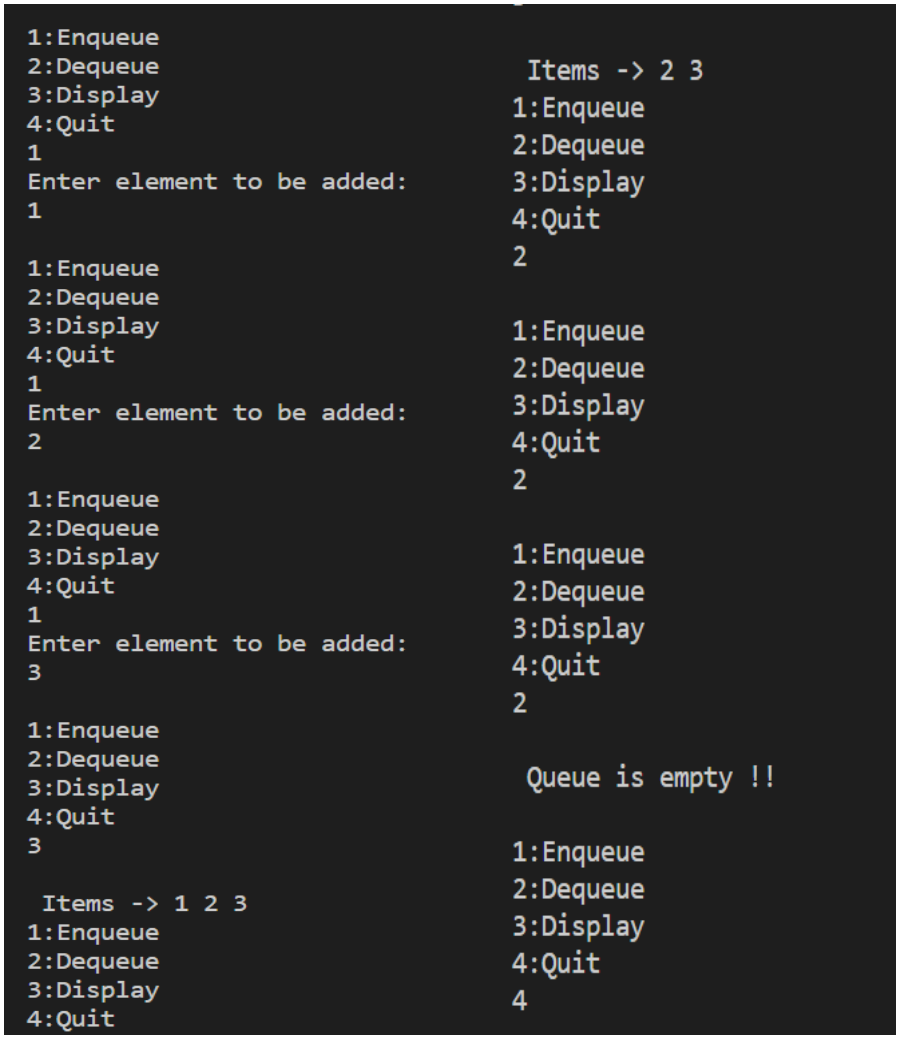
queue overflow conditions.

```
//code
// Circular Queue implementation in C

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

#define SIZE 5

int items[SIZE];
int front = -1, rear = -1;

int isFull() {
  if ((front == rear + 1) || (front == 0 && rear == SIZE - 1)) return 1;
  return 0;
}
```

```
int isEmpty() {
  if (front == -1) return 1;
  return 0;
}

void enQueue(int element) {
  if (isFull())
    printf("\n Queue is full!! \n");
  else {
    if (front == -1) front = 0;
    rear = (rear + 1) % SIZE;
    items[rear] = element;

  }
}

int deQueue() {
  int element;
  if (isEmpty()) {
    printf("\n Queue is empty !! \n");
    return (-1);
  } else {
    element = items[front];
    if (front == rear) {
      front = -1;
      rear = -1;
    }

    else {
```

```
    front = (front + 1) % SIZE;
   }

   return (element);
  }
}

void display() {
  int i;
  if (isEmpty())
   printf(" \n Empty Queue\n");
  else {

   printf("\n Items -> ");
   for (i = front; i != rear; i = (i + 1) % SIZE) {
    printf("%d ", items[i]);
   }
   printf("%d ", items[i]);
}
}

int main() {

  int choice, input;

   while (1){
      printf ("\n1:Enqueue\n2:Dequeue\n3:Display\n4:Quit\n");
      scanf ("%d",&choice);

      switch (choice){
```

```
            case 1:printf ("Enter element to be added:\n");
            scanf ("%d", &input);
            enQueue (input);
            break;
           case 2:deQueue();
            break;
           case 3:display();
            break;
           case 4:exit (0);
            break;
         default:printf ("Enter a valid choice!\n");
            break;
        }}
  return 0;
}
```


1:Enqueue
2:Dequeue
3:Display
4:Quit
1
Enter element to be added:
1
1:Enqueue
2:Dequeue
3:Display
4:Quit
1
Enter element to be added:
2
1:Enqueue
2:Dequeue
3:Display
4:Quit
1
Enter element to be added:
3
1:Enqueue
2:Dequeue
3:Display
4:Quit
3
Items -> 1 2 3
1:Enqueue
2:Dequeue
3:Display
4:Quit
Items -> 2 3
1:Enqueue
2:Dequeue
3:Display
4:Quit
2
1:Enqueue
2:Dequeue
3:Display
4:Quit
2
1:Enqueue
2:Dequeue
3:Display
4:Quit
2
Queue is empty !!
1:Enqueue
2:Dequeue
3:Display
4:Quit
4

LAB-PROGRAM-5

WAP to Implement Singly Linked List with following operations

a) Create a linked list.

b) Insertion of a node at first position, at any position and at

end of list.

c) Display the contents of the linked list.

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

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

struct node *create_node() {
   struct node *temp; int data1;
   temp=(struct node *)malloc(sizeof(struct node));
   printf("enter data: ");
   scanf("%d",&data1);
   temp->data=data1;
   temp->next=NULL;

}

void insert_beg(struct node **head) {
   struct node *ptr;
```

```
    ptr=create_node();
    ptr->next=*head;
    *head=ptr;
}

void insert_end(struct node **head) {
    struct node *ptr,*temp;
    ptr=*head;
    while((ptr->next)!=NULL) {
        ptr=ptr->next;
    }
    temp=create_node();
    ptr->next=temp;
}

void insert_mid(struct node **head,int i) {
    if(i==1) {
        insert_beg(head);
    }
    else {
    struct node *ptr,*temp;
    ptr=*head;
    temp=create_node();
    for(int j=0;j<i-2;j++) {
        ptr=ptr->next;
    }
    temp->next=ptr->next;
    ptr->next=temp;
    }
}
```

```
void display(struct node *head) {
   struct node *ptr;
   ptr=head;
   if(ptr==NULL) {
      printf("\nEmpty Linked List");
   }
   else {
      printf("\nValues: ");
      while(ptr!=NULL) {
         printf(" %d ",ptr->data);
         ptr=ptr->next;
      }
   }
}



void main() {
   struct node *head=NULL;
   while(1) {
      printf("\n1.Insert Beg\n2.Insert Mid\n3.Insert End \n4.Display\n5.Exit\n");
      int c;
      scanf("%d",&c);

      if(c==1) {
         insert_beg(&head);
      }
      else if(c==2) {
         printf("where do u want to add a node?: ");
```

```
            int pos;
            scanf("%d",&pos);
            insert_mid(&head,pos);
        }
        else if(c==3) {
            insert_end(&head);
        }
        else if(c==4) {
            display(head);
        }



        else {
            exit(0);
        }

    }
}
```

```
1.Insert Beg
2.Insert Mid
3.Insert End
4.Display
5.Exit
1
enter data: 1

1.Insert Beg
2.Insert Mid
3.Insert End
4.Display
5.Exit
1
enter data: 2

1.Insert Beg
2.Insert Mid
3.Insert End
4.Display
5.Exit
4

Values:  2  1  3
1.Insert Beg
2.Insert Mid
3.Insert End
4.Display
5.Exit
2
```

```
1.Insert Beg
2.Insert Mid
3.Insert End
4.Display
5.Exit
4

Values:  2  1  3
1.Insert Beg
2.Insert Mid
3.Insert End
4.Display
5.Exit
2
where do u want to add a node?: 1
enter data: 6

1.Insert Beg
2.Insert Mid
3.Insert End
4.Display
5.Exit
4

Values:  6  2  1  3
1.Insert Beg
2.Insert Mid
3.Insert End
4.Display
5.Exit
5

C:\Users\amshu\OneDrive\Desktop\FOLDER\codes>
```

LAB-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.

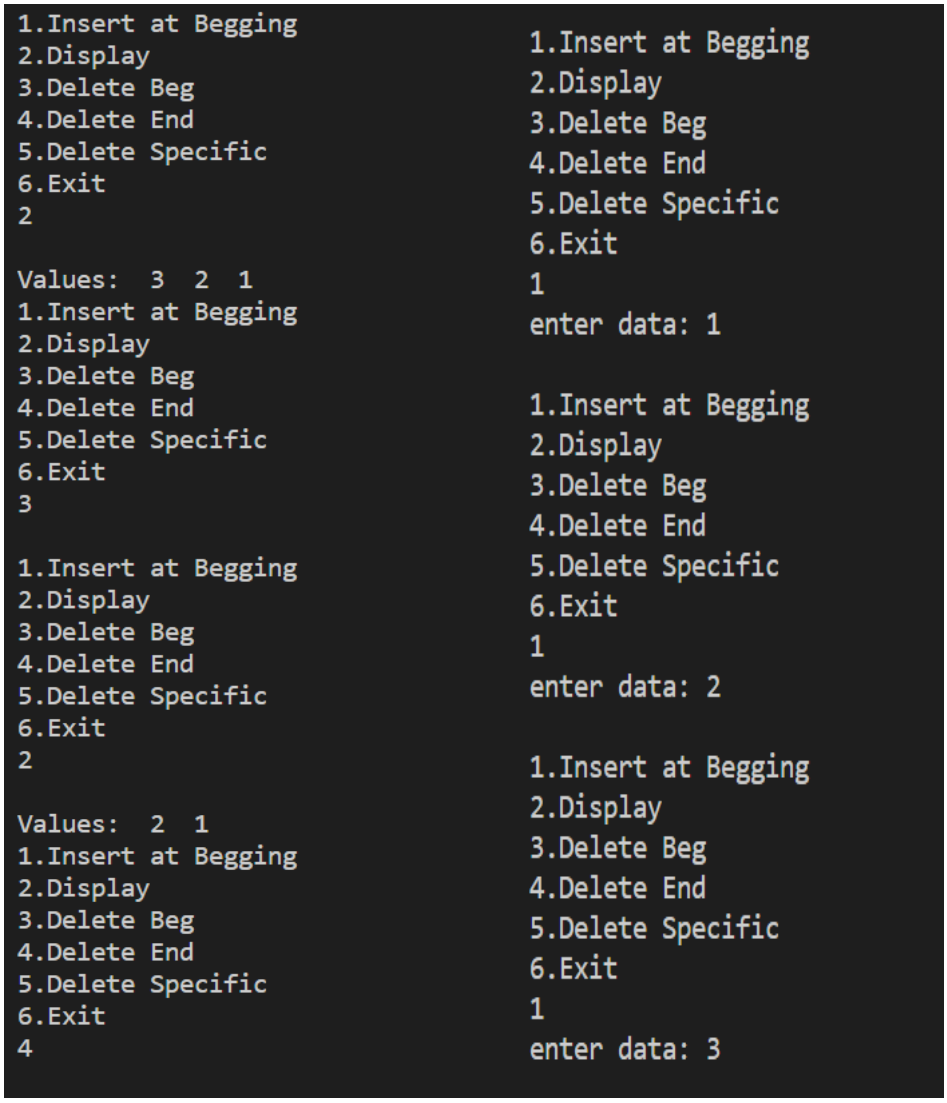
```
//code
#include <stdio.h>
#include <stdlib.h>

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

struct node *create_node() {
   struct node *temp; int data1;
   temp=(struct node *)malloc(sizeof(struct node));
   printf("enter data: ");
   scanf("%d",&data1);
   temp->data=data1;
   temp->next=NULL;

}
```

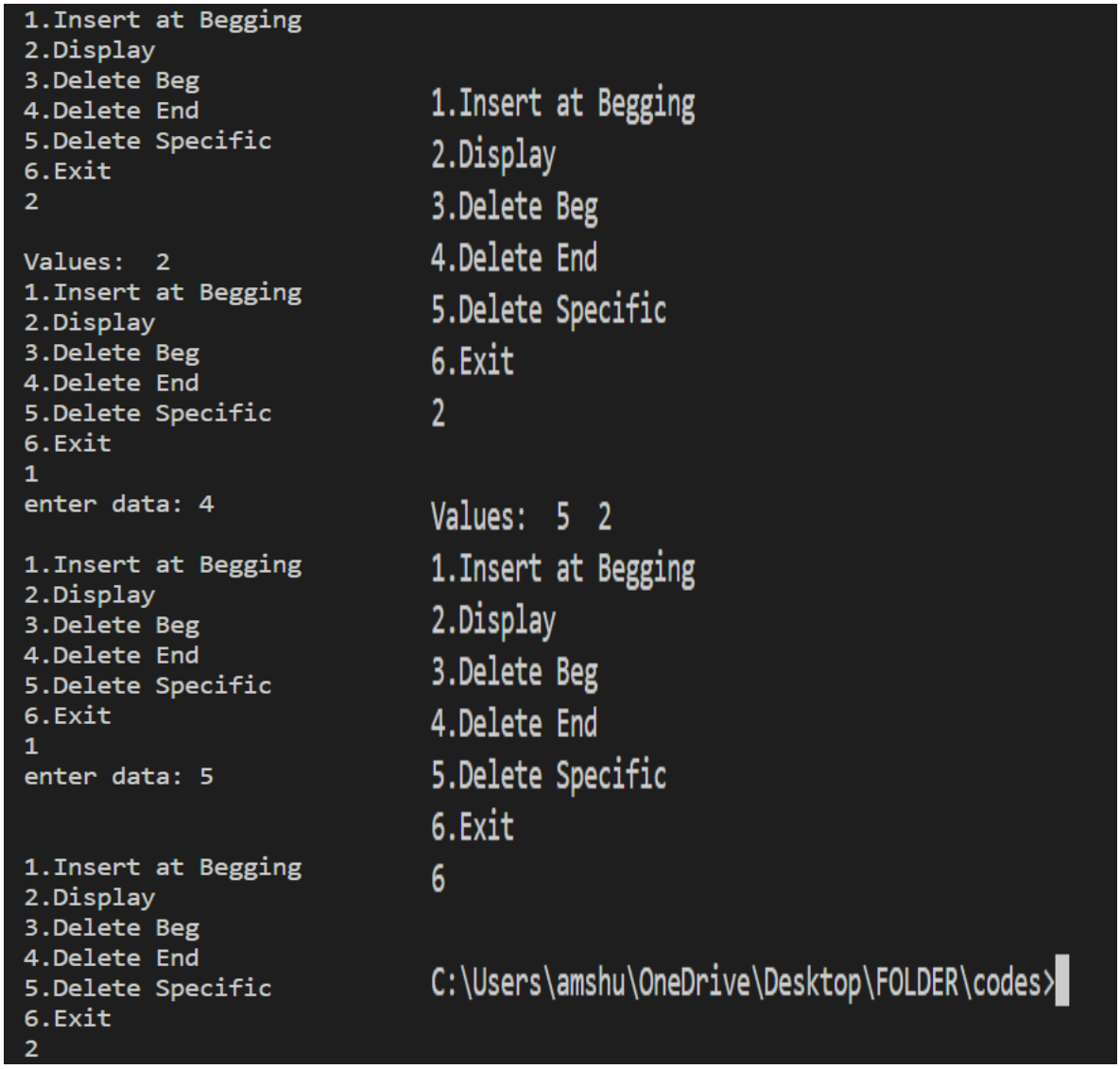
```
void insert_beg(struct node **head) {
   struct node *ptr;
   ptr=create_node();
   ptr->next=*head;
   *head=ptr;
}

void display(struct node *head) {
   struct node *ptr;
   ptr=head;
   if(ptr==NULL) {
      printf("\nEmpty Linked List");
   }
   else {
      printf("\nValues: ");
      while(ptr!=NULL) {
         printf(" %d ",ptr->data);
         ptr=ptr->next;
      }
   }
}

void delete_beg(struct node **head) {
   struct node *ptr,*temp;
   if(*head==NULL) {
      printf("List is Empty\n");
   }
   else {
      ptr=*head;
      *head=(*head)->next;
```

```
        free(ptr);
    }
}

void delete_end(struct node **head) {
    struct node *ptr, *temp;
    if(*head==NULL) {
        printf("List is Empty\n");
    }
    else if((*head)->next==NULL) {
        free(*head);
        *head=NULL;
    }
    else {
        ptr=*head;
        temp=*head;
        while((temp->next)!=NULL) {
            ptr=temp;
            temp=temp->next;
        }
        free(temp);
        ptr->next=NULL;
    }
}

void delete_specific(struct node **head,int pos) {
    struct node *ptr, *temp;
    if(*head==NULL) {
        printf("List is Empty\n");
    }
```

```
    else if((*head)->next==NULL) {
        free(*head);
        *head=NULL;
    }
    else {
        ptr=*head;
        temp=*head;
        for(int i=0;i<pos-1;i++) {
            ptr=temp;
            temp=temp->next;
        }
        ptr->next=temp->next;
        free(temp);
    }
}


void main() {
    struct node *head=NULL;
    while(1) {
        printf("\n1.Insert at Begging\n2.Display\n3.Delete Beg \n4.Delete End\n5.Delete Specific\n6.Exit\n");
        int c;
        scanf("%d",&c);

        if(c==1) {
            insert_beg(&head);
        }

        else if(c==2) {
            display(head);
```

```
        }
        else if(c==3) {
            delete_beg(&head);
        }
        else if(c==4) {
            delete_end(&head);
        }
        else if(c==5) {
            printf("where do u want to delete a node?: ");
            int pos1;
            scanf("%d",&pos1);
            delete_specific(&head,pos1);
        }

        else {
            exit(0);
        }

    }
}
```


1.Insert at Begging
2.Display
3.Delete Beg
4.Delete End
5.Delete Specific
6.Exit
2
Values: 3 2 1
1.Insert at Begging
2.Display
3.Delete Beg
4.Delete End
5.Delete Specific
6.Exit
3
1.Insert at Begging
2.Display
3.Delete Beg
4.Delete End
5.Delete Specific
6.Exit
2
Values: 2 1
1.Insert at Begging
2.Display
3.Delete Beg
4.Delete End
5.Delete Specific
6.Exit
4
1.Insert at Begging
2.Display
3.Delete Beg
4.Delete End
5.Delete Specific
6.Exit
1
enter data: 1
1.Insert at Begging
2.Display
3.Delete Beg
4.Delete End
5.Delete Specific
6.Exit
1
enter data: 2
1.Insert at Begging
2.Display
3.Delete Beg
4.Delete End
5.Delete Specific
6.Exit
1
enter data: 3


1.Insert at Begging
2.Display
3.Delete Beg
4.Delete End
5.Delete Specific
6.Exit
2
Values: 2
1.Insert at Begging
2.Display
3.Delete Beg
4.Delete End
5.Delete Specific
6.Exit
1
enter data: 4
1.Insert at Begging
2.Display
3.Delete Beg
4.Delete End
5.Delete Specific
6.Exit
1
enter data: 5
1.Insert at Begging
2.Display
3.Delete Beg
4.Delete End
5.Delete Specific
6.Exit
2
1.Insert at Begging
2.Display
3.Delete Beg
4.Delete End
5.Delete Specific
6.Exit
2
Values: 5 2
1.Insert at Begging
2.Display
3.Delete Beg
4.Delete End
5.Delete Specific
6.Exit
6
C:\Users\amshu\OneDrive\Desktop\FOLDER\codes>

LAB-PROGRAM-7

WAP to Implement Single Link List with following operations

a) Sort the linked list.

b) Reverse the linked list.

c) Concatenation of two linked lists

//CODE

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

struct NODE
{
    int value;
    struct NODE *next;
};
typedef struct NODE *node;

node insert_at_beginning(int item, node first)
{
    node temp = (node)malloc(sizeof(struct NODE));
    if (temp == NULL)
    {
        printf("\nMemory not allocated!");
    }
    (temp->value) = item;
    (temp->next) = NULL;
```

```
    if (first == NULL)
    {
        return temp;
    }
    else
    {
        temp->next = first;
        first  = temp;
        return first;
    }
}

node delete_at_the_beginning(node first)
{
    if (first == NULL)
    {
        printf("Cannot delete, the Linked List is empty");
        return NULL;
    }
    else
    {
        node temp;
        temp = first;
        first = (first->next);
        free(temp);
        return first;
    }
}
```

```
node sort(node first)
{
    int temp;
    node curr = first;
    if (first == NULL)
    {
        printf("Linked list is empty!");
        return NULL;
    }
    else
    {
        while (curr->next != NULL)
        {
            node check = curr->next;
            while (check != NULL)
            {
                if (curr->value > check->value)
                {
                    temp = curr->value;
                    curr->value = check->value;
                    check->value = temp;
                }
                check = check->next;
            }
            curr = curr->next;
        }
        return first;
```

```
    }
}

node concatenate(node f1, node f2)
{
    if (f1 == NULL && f2 == NULL)
    {
        printf("The linked lists are empty!");
        return NULL;
    }
    else if (f1 != NULL && f2 == NULL)
        return f1;
    else if (f1 == NULL && f2 != NULL)
        return f2;
    else
    {
        node last = f1;
        while (last->next != NULL)
        {
            last = last->next;
        }
        last->next = f2;
        return f1;
    }
}

node reverse(node first)
{
```

```
    if (first == NULL)
    {
        printf("The linked lists are empty!");
        return NULL;
    }
    else
    {
        node rev = NULL;
        while (first != NULL)
        {
            node Next = first->next;
            first->next = rev;
            rev = first;
            first = Next;
        }
        return rev;
    }
}

void display(node first)
{
    node temp;
    temp = first;
    if (temp == NULL)
    {
        printf("The Linked list is empty!");
    }
    else
```

```
    {
        printf("The elements in the node are :  ");
        while (temp != NULL)
        {
            printf("%d  ", (temp->value));
            temp = (temp->next);
        }
    }
}

int main()
{
    int choice, n, i, val, x;
    node first = NULL, f1 = NULL, f2 = NULL;
    while (1)
    {
        printf("\n\nEnter the operations to be performed :\n1) Push\n2) Pop\n3) Sort\n4) Concatenate\n5) Reverse\n6) Display\nEnter your choice : ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("Enter the element to be inserted : ");
            scanf("%d", &x);
            first = insert_at_beginning(x, first);
            break;
        case 2:
            first = delete_at_the_beginning(first);
            break;
```

```
case 3:
    first = sort(first);
    break;
case 4:

    printf("Enter the number of fields for linked list 1 : ");
    scanf("%d", &n);
    printf("Enter %d entries : ", n);
    for (i = 0; i < n; i++)
    {
        scanf("%d", &val);
        f1 = insert_at_beginning(val, f1);
    }
    printf("Enter the number of fields for linked list 2 : ");
    scanf("%d", &n);
    printf("Enter %d entries : ", n);
    for (i = 0; i < n; i++)
    {
        scanf("%d", &val);
        f2 = insert_at_beginning(val, f2);
    }
    printf("The concatenated linked list is : ");
    f1 = concatenate(f1, f2);
    display(f1);
    break;
case 5:
    first = reverse(first);
    break;
```

```
        case 6:
            display(f
            irst);
            break;
        default:
            exit(0);
        }
    }
    return 0;
}
```

```
Enter the operations to be performed :
1) Push
2) Pop
3) Sort
4) Concatenate
5) Reverse
6) Display
Enter your choice : 1
Enter the element to be inserted : 1


Enter the operations to be performed :
1) Push
2) Pop
3) Sort
4) Concatenate
5) Reverse
6) Display
Enter your choice : 1
Enter the element to be inserted : 0


Enter the operations to be performed :
1) Push
2) Pop
3) Sort
4) Concatenate
5) Reverse
6) Display
Enter your choice : 1
Enter the element to be inserted : 7
```

```
Enter the operations to be performed :
1) Push
2) Pop
3) Sort
4) Concatenate
5) Reverse
6) Display
Enter your choice : 1
Enter the element to be inserted : 9


Enter the operations to be performed :
1) Push
2) Pop
3) Sort
4) Concatenate
5) Reverse
6) Display
Enter your choice : 6
The elements in the node are : 9 7 0 1

Enter the operations to be performed :
1) Push
2) Pop
3) Sort
4) Concatenate
5) Reverse
6) Display
Enter your choice : 5
```

```
Enter the operations to be performed :
1) Push
2) Pop
3) Sort
4) Concatenate
5) Reverse
6) Display
Enter your choice : 6
The elements in the node are : 1 0 7 9

Enter the operations to be performed :
1) Push
2) Pop
3) Sort
4) Concatenate
5) Reverse
6) Display
Enter your choice : 6
The elements in the node are : 0 1 7 9
```

LAB-PROGRAM-8

WAP to implement Stack & Queues using Linked Representation

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

struct NODE
{
   int value;
   struct NODE *next;
};
typedef struct NODE *node;

node insert_at_beginning(int item, node first)
{
   node temp = (node)malloc(sizeof(struct NODE));
   if (temp == NULL)
   {
      printf("\nMemory not allocated!");
   }
   (temp->value) = item;
   (temp->next) = NULL;
   if (first == NULL)
   {
      return temp;
   }
   else
```

```
    {
        temp->next = first;
        first = temp;
        return first;
    }
}

node delete_at_the_beginning(node first)
{
    if (first == NULL)
    {
        printf("\nCannot delete, the Linked List is empty");
        return NULL;
    }
    else
    {
        node temp;
        temp = first;
        first = (first->next);
        free(temp);
        return first;
    }
}

node delete_at_the_end(node first)
{
    if (first == NULL)
    {
        printf("\nCannot delete, the Linked List is empty");
```

```
        return NULL;
    }
    else
    {
        node prev, curr;
        prev = NULL;
        curr = first;
        while ((curr->next) != NULL)
        {
            prev = curr;
            curr = (curr->next);
        }
        (prev->next) = NULL;
        free(curr);
        return first;
    }
}

void display(node first)
{
    node temp;
    temp = first;
    if (temp == NULL)
    {
        printf("\nThe Linked list is empty!");
    }
    else
    {
        printf("The elements in the node are : ");
```

```
        while (temp != NULL)
        {
            printf("%d\t", (temp->value));
            temp = (temp->next);
        }
    }
}

int main()
{
    int choice, pos, item, x;
    node first = NULL;
    printf("Enter the data structure you would want to create :\n1) Stack\n2) Queue\nYour choice : ");
    scanf("%d", &choice);
    if (choice == 1)
    {
        while (1)
        {
            printf("\n\nEnter the operations to be performed :\n1) Push\n2) Pop\n3) Display\nEnter your
choice : ");
            scanf("%d", &choice);
            switch (choice)
            {
            case 1:
                printf("Enter the element to be inserted : ");
                scanf("%d", &x);
                first = insert_at_beginning(x, first);
                break;
            case 2:
```

```
            first = delete_at_the_beginning(first);
            break;
        case 3:
            display(first);
            break;
        default:
            exit(0);
        }
    }
  }
  else if (choice == 2)
  {
    while (1)
    {
        printf("\n\nEnter the operations to be performed :\n1) Enqueue\n2) Dequeue\n3) Display\nEnter your choice : ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("Enter the element to be inserted : ");
            scanf("%d", &x);
            first = insert_at_beginning(x, first);
            break;
        case 2:
            first = delete_at_the_end(first);
            break;
        case 3:
            display(first);
```

```
            break;
        default:
            exit(0);
        }
    }
  }
  else
  {
    printf("Enter a valid choice!");
  }
  return 0;
}
```

```
Enter the data structure you would want to create :
1) Stack
2) Queue
Your choice : 1


Enter the operations to be performed :
1) Push
2) Pop
3) Display
Enter your choice : 1
Enter the element to be inserted : 1


Enter the operations to be performed :
1) Push
2) Pop
3) Display
Enter your choice : 1
Enter the element to be inserted : 2


Enter the operations to be performed :
1) Push
2) Pop
3) Display
Enter your choice : 3
The elements in the node are : 2        1

Enter the operations to be performed :
1) Push
2) Pop
3) Display
Enter your choice : 2
```

```
 Enter the operations to be performed :
 1) Push
 2) Pop
 3) Display
 Enter your choice : 2


 Enter the operations to be performed :
 1) Push
 2) Pop
 3) Display
 Enter your choice : 3
 The elements in the node are : 1

 Enter the operations to be performed :
 1) Push
 2) Pop
 3) Display
 Enter your choice : 4
```

```
Enter the data structure you would want to create :
1) Stack
2) Queue
Your choice : 2


Enter the operations to be performed :
1) Enqueue
2) Dequeue
3) Display
Enter your choice : 1
Enter the element to be inserted : 1


Enter the operations to be performed :
1) Enqueue
2) Dequeue
3) Display
Enter your choice : 1
Enter the element to be inserted : 2


Enter the operations to be performed :
1) Enqueue
2) Dequeue
3) Display
Enter your choice : 1
Enter the element to be inserted : 3


Enter the operations to be performed :
1) Enqueue
2) Dequeue
3) Display
Enter your choice : 3
```

```
Enter the operations to be performed :
1) Enqueue
2) Dequeue
3) Display
Enter your choice : 3
The elements in the node are : 3        2       1

Enter the operations to be performed :
1) Enqueue
2) Dequeue
3) Display
Enter your choice : 2


Enter the operations to be performed :
1) Enqueue
2) Dequeue
3) Display
Enter your choice : 3
The elements in the node are : 3        2
```

LAB-PROGRAM-9

WAP to Implement doubly link list with primitive operations

a) Create a doubly linked list.

b) Insert a new node to the left of the node.

c) Delete the node based on a specific value

d) Display the contents of the list

```
//CODE#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct Node{
    int value;
    struct Node *next;
    struct Node *prev;
};

typedef struct Node *Node;

Node getNode(){

    Node temp;
    temp=(Node)malloc(sizeof(struct Node));

    if(temp==NULL){
        printf("Memory not allocated\n");
    }
    return temp;
```

```
}

Node insert_beg(Node first,int item){
    Node new;
    new=getNode();
    new->value=item;
    new->prev=NULL;
    new->next=NULL;
    if(first==NULL){
        return new;
    }
    new->next=first;
    first->prev=new;
    return new;
}

Node insert_left(Node first,int key,int item){

   Node temp,new;

   new=getNode();
   new->value=item;
   new->prev=NULL;
   new->next=NULL;

   if(first==NULL){
      printf("List is empty");
      return NULL;
   }
```

```
if (first->value==key){
   first=insert_beg(first, item);
   return first;
}

if(first->next==NULL && first->value!=key){
   printf("Elements does not exist!\n");
   return first;
}

if(first->next==NULL && first->value==key){
   first=insert_beg(first,new->value);
}
temp=first;

while(temp->value!=key && temp->next!=NULL){
   temp=temp->next;
}
if(temp->value==key){
   new->next=temp;
   new->prev=temp->prev;
   (temp->prev)->next=new;
   temp->prev=new;
   return first;
}
if(temp->value!=key){
    printf("value not found\n");
   return first;
}
```

```
}

Node delete_specific(Node first,int key){
    Node curr,temp;
    curr=first;
    if(first==NULL){
        printf("list empty!\n");
        return NULL;
    }
    if(first->next==NULL && first->value==key){
        free(first);
        return NULL;
    }
    if(first->value==key){
        (first->next)->prev=NULL;
        temp=first->next;
        free(first);
        return temp;
    }
    while(curr!=NULL){
        if(curr->value==key)
        break;
        curr=curr->next;
    }
    if(curr==NULL){
        printf("Element not found\n");
    }
    (curr->prev)->next=curr->next;
```

```
    if(curr->next!=NULL){
        (curr->next)->prev=curr->prev;
    }
    return first;
}

void display(Node first){
    Node temp;

    if(first==NULL){
        printf("List is empty\n");
    }
    temp=first;
    while(temp!=NULL){
        printf("%d  ",temp->value);
        temp=temp->next;
    }
}

void main(){
   Node first=NULL;
   int choice,key,item;
   while(1){

   printf("\n1.Insert\n2.Insert left\n3.Delete\n4.Display\n5.Exit\n");
   scanf("%d",&choice);

   switch(choice){
   case 1:printf("\nEnter the value to be inserted at the begining\n");
```

```
        scanf("%d",&item);
        first=insert_beg(first,item);
        break;

    case 2:printf("\nEnter the value to be inserted at the left\n");
        scanf("%d",&item);
        printf("\nEnter element before\n");
        scanf("%d",&key);
        first=insert_left(first,key,item);
        break;

    case 3:printf("\nEnter the value to be deleted\n");
       scanf("%d",&key);
       first=delete_specific(first,key);
       break;

    case 4:display(first);
       break;

    case 5:exit(0);
    break;

    default:exit(0);
     }
     }
}
```

```
1.Insert
2.Insert left
3.Delete
4.Display
5.Exit
1

Enter the value to be inserted at the begining
1

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

Enter the value to be inserted at the begining
3

1.Insert
2.Insert left
3.Delete
4.Display
5.Exit
4
3  1
1.Insert
2.Insert left
3.Delete
4.Display
Enter element before
3
```

```
4
3  1
1.Insert
2.Insert left
3.Delete
4.Display
Enter element before
3

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

Enter the value to be deleted
3

1.Insert
2.Insert left
3.Delete
4.Display
5.Exit
4
3  1
1.Insert
2.Insert left
3.Delete
4.Display
5.Exit
5
```

LAB-PROGRAM-10

Write a program

a) To construct a binary Search tree.

b) To traverse the tree using all the methods i.e., in-order, preorder and post order

c) To display the elements in the tree.

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

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

struct node *insert(struct node *node, int data)
{
   if (node == NULL)
   {
      struct node *temp = (struct node *)malloc(sizeof(struct node));
      temp->data = data;
      temp->left = temp->right = NULL;
```

```
        return temp;
    }

    if (data < node->data)
        node->left = insert(node->left, data);
    else if (data > node->data)
        node->right = insert(node->right, data);

    return node;
}

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);
   }
}

int main()
{
   struct node *root = NULL;
   int n, i, element;

   printf("Enter the number of elements to be inserted: ");
   scanf("%d", &n);
   printf("Enter %d elements: ", n);
   for (i = 0; i < n; i++)
   {
      scanf("%d", &element);
      root = insert(root, element);
   }

   printf("In-order traversal: ");
   inorder(root);
   printf("\nPre-order traversal: ");
```

```
    preorder(root);
    printf("\nPost-order traversal: ");
    postorder(root);

    return 0;
}
```

```
Enter the number of elements to be inserted: 5
Enter 5 elements: 1 6 4 7 5
In-order traversal: 1 4 5 6 7
Pre-order traversal: 1 6 4 5 7
Post-order traversal: 5 4 7 6 1
```