

**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**  
“JnanaSangama”, Belgaum -590014, Karnataka.



**LAB REPORT**  
on  
**DATA STRUCTURES**

*Submitted by*

**Pulkit Raina (1BM21CS148)**

*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 **Pulkit Raina (1BM21CS148)**, 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.

Sheetal VA  
Assistant Professor  
Department of CSE  
BMSCE, Bengaluru

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

# Index Sheet

<b>Sl. No.</b>	<b>Experiment Title</b>	<b>Pg No.</b>
1.	<p>Write a program to simulate the working of stack using an array with the following:</p> <ul style="list-style-type: none"> <li>a) Push</li> <li>b) Pop</li> <li>c) Display</li> </ul> <p>The program should print appropriate messages for stack overflow, stack underflow</p>	5
2.	<p>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)</p>	8
3.	<p>WAP to simulate the working of a queue of integers using an array. Provide the following operations</p> <ul style="list-style-type: none"> <li>a) Insert</li> <li>b) Delete</li> <li>c) Display</li> </ul> <p>The program should print appropriate messages for queue empty and queue overflow conditions</p>	11
4.	<p>WAP to simulate the working of a circular queue of integers using an array. Provide the following operations.</p> <ul style="list-style-type: none"> <li>a) Insert</li> <li>b) Delete</li> <li>c) Display</li> </ul> <p>The program should print appropriate messages for queue empty and queue overflow conditions</p>	14
5.	<p>WAP to Implement Singly Linked List with following operations</p> <ul style="list-style-type: none"> <li>a) Create a linked list.</li> <li>b) Insertion of a node at first position, at any position and at end of list.</li> <li>c) Display the contents of the linked list.</li> </ul>	18
6.	<p>WAP to Implement Singly Linked List with following operations</p> <ul style="list-style-type: none"> <li>a) Create a linked list.</li> <li>b) Deletion of first element, specified element and last element in the list.</li> <li>c) Display the contents of the linked list.</li> </ul>	21
7.	<p>WAP to Implement Single Link List with following operations</p> <ul style="list-style-type: none"> <li>a) Sort the linked list.</li> <li>b) Reverse the linked list.</li> <li>c) Concatenation of two linked lists</li> </ul>	27

8.	WAP to implement Stack & Queues using Linked Representation	32
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	39
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.	45

## 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:**

```
33
34 void push(int stk[], int n, int *top)
35 {
36     int x;
37     if (*top!=n-1)
38     {
39         printf("Enter the element to push in: ");
40         scanf("%d", &x);
41         stk[++(*top)] = x;
42     }
43     else
44         printf("STACK OVERFLOW\\n");
45 }
46
47 void pop(int stk[], int *top)
48 {
49     if (*top<0)
50         printf("STACK UNDERFLOW\\n");
51     else
52     {
53         printf("Popped element is %d\\n", stk[*top]);
54         (*top)--;
55     }
56 }
57
58 void display(int stk[], int *top)
59 {
60     int i;
61     if (*top== -1)
62         printf("STACK UNDERFLOW\\n");
63     else
64     {
65         printf("Stack is:\\n");
66         for (i= *top; i>-1; i--)
67             printf("%d\\n", stk[i]);
68     }
69 }
70 }
```

**OUTPUT:**

```
Enter the size of stack: 3
Enter your choice:
1. Push into Stack
2. Pop from Stack
3. Display Stack
4. Exit
1
Enter the element to push in: 1
1
Enter the element to push in: 2
1
Enter the element to push in: 3
1
STACK OVERFLOW
3
Stack is:
3
2
1
2
Popped element is 3
2
Popped element is 2
2
Popped element is 1
2
STACK UNDERFLOW
3
STACK UNDERFLOW
4

Process returned 0 (0x0)  execution
Press any key to continue.
```

## 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:

```
1 #include<stdio.h>
2 #include<ctype.h>
3 #include<string.h>
4
5 void infixToPostfix(char in[]);
6 int getPriority(char c);
7
8 int main() {
9     char in[100];
10    int l;
11    printf("Enter the infix expression: ");
12    gets(in);
13    l = strlen(in);
14    in[l] = '\n';
15    infixToPostfix(in);
16    return 0;
17 }
18
19
20
21 int getPriority(char c) {
22     if(c == '*' || c == '/')
23         return 9;
24     else if(c == '+' || c == '-')
25         return 8;
26     else
27         return -1;
28 }
```

```
29
30 void infixToPostfix(char in[]){
31     char out[100];
32     int stk[100];
33     int i = 0, j = -1, top = -1;
34     while(in[i] != '\n'){
35         if(isalnum(in[i]))
36             out[++j] = in[i];
37
38         else if(in[i] == '(')
39             stk[++top] = in[i];
40
41         else if(in[i] == '+' || in[i] == '-' || in[i] == '*' ||
42                  in[i] == '/'){
43             while(getPriority(in[i]) <= getPriority(stk[top])){
44                 out[++j] = stk[top--];
45                 stk[++top] = in[i];
46             }
47         else if(in[i] == ')'){
48             while(stk[top] != '('){
49                 out[++j] = stk[top--];
50             }
51             top--;
52         }
53         else{
54             printf("INCORRECT EXPRESSION\n");
55             return;
56         }
57         i++;
58     }
59     while(top != -1)
60         out[++j] = stk[top--];
61     printf("%s", out);
62 }
63 }
```

**OUTPUT:**

```
Enter the infix expression: A-(B/C+(D/E*F)/G)*H  
ABC/DE/F*G/+H*-  
Process returned 0 (0x0) execution time : 16.8  
Press any key to continue.
```

```
Enter the infix expression: (A+B)*(C+D)  
AB+CD+*  
Process returned 0 (0x0) execution time :  
Press any key to continue.
```

## 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:

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #define MAX 3
4
5 int main()
6 {
7     int queue[100], front = -1, rear = -1;
8     int n,value,choice;
9     printf("Enter length of queue(MAX 100): ");
10    scanf("%d", &n);
11    printf("Choose : \n1.INSERT\n2.DELETE\n3.DISPLAY\n4.EXIT\n");
12    while(1)
13    {
14        scanf("%d",&choice);
15        switch(choice)
16        {
17            case 1:
18                enqueue(queue, &front, &rear, n);
19                break;
20            case 2:
21                dequeue(queue, &front, &rear);
22                break;
23            case 3:
24                display(queue, &front, &rear);
25                break;
26            case 4:
27                exit(0);
28            default:
29                printf("INCORRECT INPUT TRY AGAIN");
30
31        }
32    }
33 }
34 }
```

```
50
36 void enqueue(int queue[], int *front, int *rear, int n)
37 {
38     int value;
39     if(*rear==n-1)
40         printf("OVER FLOW\n");
41     else
42     {
43         printf("Enter value to be inserted: ");
44         scanf("%d", &value);
45         if (*front== -1 && *rear== -1)
46             *front=0;
47         queue[+>(*rear)]=value;
48     }
49 }
50
51 void dequeue(int queue[], int *front, int *rear)
52 {
53     if(*front== -1 || *front>*rear)
54         printf("UNDER FLOW\n");
55     else
56     {
57         printf("value deleted : %d\n", queue[*front]);
58         (*front)++;
59     }
60 }
61
62
63 void display(int queue[], int *front, int *rear)
64 {
65     int i;
66     if(*front>*rear || *front== -1 && *rear == -1)
67         printf("UNDERFLOW\n");
68     else
69     {
70         for(i=*front;i<=*rear;i++)
71             printf("%d\n", queue[i]);
72     }
}
```

**OUTPUT:**

```
Enter Length of queue(MAX 100): 5
Choose :
1. INSERT
2.DELETE
3.DISPLAY
4.EXIT
1
Enter value to be inserted: 1
1
Enter value to be inserted: 2
1
Enter value to be inserted: 3
1
Enter value to be inserted: 4
1
Enter value to be inserted: 5
1
OVER FLOW
3
1
2
3
4
5
2
value deleted : 1
2
value deleted : 2
2
value deleted : 3
2
value deleted : 4
2
value deleted : 5
2
UNDER FLOW
3
UNDERFLOW
1
OVER FLOW
4

Process returned 0 (0x0)  execution
Press any key to continue.
```

## 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:

```
1 #include <stdio.h>
2
3 int isFull(int front, int rear, int n);
4 int isEmpty(int front);
5 void enqueue(int items[], int *front, int *rear, int n);
6 int dequeue(int items[], int *front, int *rear, int n);
7 void display(int items[], int *front, int *rear, int n);
8
9
10 int main()
11 {
12     int items[100], front = -1, rear = -1, n, choice;
13     printf("Enter size of queue(max is 100): ");
14     scanf("%d", &n);
15     printf("Enter your choice:\n1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");
16     while(1)
17     {
18         scanf("%d", &choice);
19         switch (choice)
20         {
21             case 1:
22                 enqueue(items, &front, &rear, n);
23                 break;
24             case 2:
25                 dequeue(items, &front, &rear, n);
26                 break;
27             case 3:
28                 display(items, &front, &rear, n);
29                 break;
30             case 4:
31                 return 0;
32             default:
33                 printf("Incorrect Choice! Enter again:\n");
34                 break;
35         }
36     }
37     return 0;
38 }
39
40 int isFull(int front, int rear, int n)
41 {
42     if ((front == rear + 1) || (front == 0 && rear == n - 1))
43         return 1;
44     return 0;
45 }
46
47 int isEmpty(int front)
48 {
49     if (front == -1)
50         return 1;
51     return 0;
52 }
```

```

53
54 void enQueue(int items[], int *front, int *rear, int n)
55 {
56     int element;
57     if (isFull(*front, *rear, n))
58         printf("Queue is full!! \n");
59     else
60     {
61         printf("Enter element to enqueue: ");
62         scanf("%d", &element);
63         if (*front == -1)
64             *front = 0;
65         *rear = (*rear + 1) % n;
66         items[*rear] = element;
67         printf("Inserted -> %d\n", element);
68     }
69 }
70
71 int deQueue(int items[], int *front, int *rear, int n)
72 {
73     int element;
74     if (isEmpty(*front))
75     {
76         printf("Queue is empty !! \n");
77         return (-1);
78     }
79     else
80     {
81         element = items[*front];
82         if (*front == *rear)
83         {
84             *front = -1;
85             *rear = -1;
86         }
87         else
88             *front = (*front + 1) % n;
89         printf("Deleted element -> %d \n", element);
90         return (element);
91     }
92 }
93
94 void display(int items[], int *front, int *rear, int n)
95 {
96     int i;
97     if (isEmpty(*front))
98         printf("Empty Queue\n");
99     else
100    {
101        printf("Items -> ");
102        printf("%d\t", items[*front]);
103        for (i = *front+1; i != *rear; i = (i + 1) % n)
104            printf("%d\t", items[i]);
105        printf("%d\n", items[*rear]);
106    }
107 }
108

```

**OUTPUT:**

```
Enter size of queue(max is 100): 5
Enter your choice:
1. Enqueue
2. Dequeue
3. Display
4. Exit:
1
Enter element to enqueue: 1
Inserted -> 1
1
Enter element to enqueue: 2
Inserted -> 2
1
Enter element to enqueue: 3
Inserted -> 3
1
Enter element to enqueue: 4
Inserted -> 4
1
Enter element to enqueue: 5
Inserted -> 5
1
Queue is full!!
3
Items -> 1      2      3      4      5
2
Deleted element -> 1
2
Deleted element -> 2
2
Deleted element -> 3
2
Deleted element -> 4
2
Deleted element -> 5
2
Queue is empty !!
```

```
3
Empty Queue
1
Enter element to enqueue: 1
Inserted -> 1
1
Enter element to enqueue: 2
Inserted -> 2
1
Enter element to enqueue: 3
Inserted -> 3
2
Deleted element -> 1
3
Items -> 2      3
4

Process returned 0 (0x0)   execution time : 0.012 s
Press any key to continue.
```

## 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.

## CODE:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 typedef struct nd {
6     int data;
7     struct node *next;
8 }node;
9
10 node* insertBeginning(node* head);
11 node* insertEnd(node* head);
12 node* insertMiddle(node* head, int val);
13
14 int main() {
15     node* head = NULL;
16     int choice, val;
17     printf("Enter\n1. Insert Node at beginning\n2. Insert Node at end\n");
18     printf("3. Insert Node after a value\n4. Display List\n5. Exit: ");
19     while(1) {
20         scanf("%d", &choice);
21         switch(choice) {
22             case 1:
23                 head = insertBeginning(head);
24                 break;
25             case 2:
26                 head = insertEnd(head);
27                 break;
28             case 3:
29                 printf("Enter val after which you insert node: ");
30                 scanf("%d", &val);
31                 head = insertMiddle(head, val);
32                 break;
33             case 4:
34                 displayList(head);
35                 break;
36             case 5:
37                 return 0;
38             default:
39                 printf("INCORRECT VALUE!\n");
40         }
41     }
42     return 0;
43 }
44 }
```

```
45
46     node* insertBeginning(node* head) {
47         int val;
48         node* temp;
49         if(head == NULL) {
50             printf("NEW LINKED LIST CREATED!\n");
51             printf("Enter element to insert: ");
52             scanf("%d", &val);
53             head = (node*)malloc(sizeof(node));
54             head->next = NULL;
55             head->data = val;
56         }
57         else{
58             printf("Enter element to insert: ");
59             scanf("%d", &val);
60             temp = (node*)malloc(sizeof(node));
61             temp->next = head;
62             temp->data = val;
63             head = temp;
64         }
65         return head;
66     }
67
68
69     node* insertEnd(node* head) {
70         node* temp = head;
71         node* ptr;
72         int val;
73         while(temp->next!=NULL)
74             temp = temp->next;
75         ptr = (node*)malloc(sizeof(node));
76         printf("Enter val to insert: ");
77         scanf("%d", &val);
78         ptr->data = val;
79         ptr->next = NULL;
```

```
80     temp->next = ptr;
81     return head;
82 }
83
84 node* insertMiddle(node* head, int n) {
85     node* temp = head;
86     node* ptr;
87     int val;
88     while(temp->data != n && temp->next != NULL)
89         temp = temp->next;
90     if(temp->data == n) {
91         printf("Enter val to insert: ");
92         scanf("%d", &val);
93         ptr = (node*)malloc(sizeof(node));
94         ptr->next = temp->next;
95         ptr->data = val;
96         temp->next = ptr;
97     }
98     else
99         printf("Value NOT Found in List!\n");
100    return head;
101 }
102
103 void displayList(node* head) {
104     node* temp = head;
105     while(temp!=NULL) {
106         printf("%d -> ", temp->data);
107         temp = temp->next;
108     }
109     printf("NULL\n");
110 }
111 }
```

**OUTPUT:**

```
Enter
1. Insert Node at beginning
2. Insert Node at end
3. Insert Node after a value
4. Display List
5. Exit: 1
NEW LINKED LIST CREATED!
Enter element to insert: 1
1
Enter element to insert: 2
1
Enter element to insert: 3
1
Enter element to insert: 4
4
4 -> 3 -> 2 -> 1 -> NULL
2
Enter val to insert: 5
4
4 -> 3 -> 2 -> 1 -> 5 -> NULL
3
Enter val after which you insert node: 2
Enter val to insert: 7
4
4 -> 3 -> 2 -> 7 -> 1 -> 5 -> NULL
5

Process returned 0 (0x0)  execution time
Press any key to continue.
```

## 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:

```
1 #include<stdio.h>
2
3 typedef struct nd{
4     int data;
5     struct nd* next;
6 }node;
7
8 node* createList(node* head, int n);
9 node* deleteFirst(node* head);
10 node* deleteLast(node* head);
11 node* deleteSpecific(node* head, int n);
12 void display(node* head);
13
14 int main(){
15     int choice, n;
16     node* head = NULL;
17     printf("Enter\n1. Create List of n nodes\n2. Delete from front\n3. Delete from end\n");
18     printf("4. Delete specific data\n5. Display List\n6. Exit: ");
19     while(1){
20         scanf("%d", &choice);
21         switch(choice){
22             case 1:
23                 printf("Enter number of nodes to create: ");
24                 scanf("%d", &n);
25                 head = createList(head, n);
26                 break;
27             case 2:
28                 head = deleteFirst(head);
29                 break;
30             case 3:
31                 head = deleteLast(head);
32                 break;
33             case 4:
34                 printf("Enter data to delete: ");
35                 scanf("%d", &n);
36                 head = deleteSpecific(head, n);
37                 break;
38             case 5:
39                 display(head);
40                 break;
41             case 6:
42                 return 0;
43             default:
44                 printf("INCORRECT ENTRY!\n");
45         }
46     }
47 }
```

```
48 node* createList(node* head, int n){
49     node* temp;
50     node* ptr = head;
51     int val;
52     for(int i = 0; i<n; i++){
53         temp = (node*)malloc(sizeof(node));
54         if(i==0){
55             printf("New Linked List created!\n");
56             printf("Enter the data for first node: ");
57             scanf("%d", &val);
58             temp->data = val;
59             temp->next = NULL;
60             head = temp;
61             printf("NODE 1 created\n");
62         }
63         else{
64             ptr = head;
65             while(ptr->next!=NULL)
66                 ptr = ptr->next;
67             printf("Enter the data for node: ");
68             scanf("%d", &val);
69             temp->data = val;
70             temp->next = NULL;
71             ptr->next = temp;
72         }
73     }
74     return head;
75 }
76
77
78 node* deleteFirst(node* head) {
79     node* temp = head;
```

```
80     if(head == NULL)
81         printf("EMPTY LIST\n");
82     else{
83         head = head->next;
84         free(temp);
85     }
86     return head;
87 }
88
89 node* deleteLast(node* head) {
90     node* temp = head;
91     node* ptr;
92     if(head == NULL){
93         printf("EMPTY LIST\n");
94         return head;
95     }
96     else if(head->next == NULL){
97         head = NULL;
98         free(temp);
99         return head;
100    }
101   else{
102       while(temp->next->next!=NULL)
103           temp = temp->next;
104       ptr = temp->next;
105       temp->next = NULL;
106       free(ptr);
107   }
108   return head;
109 }
110 }
```

```
111 node* deleteSpecific(node* head, int val){  
112     node* temp = head;  
113     node* ptr = head;  
114     if(head == NULL){  
115         printf("EMPTY LIST\n");  
116         return head;  
117     }  
118     if(head->next == NULL){  
119         head = NULL;  
120         free(temp);  
121         return head;  
122     }  
123     while(temp->data != val && temp->next != NULL)  
124         temp = temp->next;  
125     if(temp->data == val){  
126         ptr = head;  
127         while(ptr->next!=temp)  
128             ptr = ptr->next;  
129         ptr->next = temp->next;  
130         free(temp);  
131         return head;  
132     }  
133     else{  
134         printf("NO SUCH ELEMENT\n");  
135         return head;  
136     }  
137 }  
138  
139 void display(node* head){  
140     node* temp = head;  
141     if(head == NULL){  
142         printf("EMPTY LIST\n");  
143         return;  
144     }  
145     while(temp != NULL){  
146         printf("%d -> ", temp->data);  
147         temp = temp->next;  
148     }  
149     printf("NULL\n");  
150 }
```

**OUTPUT:**

```
Enter
1. Create List of n nodes
2. Delete from front
3. Delete from end
4. Delete specific data
5. Display List
6. Exit: 5
EMPTY LIST!
4
Enter data to delete: 3
EMPTY LIST
3
EMPTY LIST
2
EMPTY LIST
1
Enter number of nodes to create: 5
New Linked List created!
Enter the data for first node: 1
NODE 1 created
Enter the data for node: 2
Enter the data for node: 3
Enter the data for node: 4
Enter the data for node: 5
5
1 -> 2 -> 3 -> 4 -> 5 -> NULL
2
5
2 -> 3 -> 4 -> 5 -> NULL
3
5
2 -> 3 -> 4 -> NULL
6

Process returned 0 (0x0)  execution time :
Press any key to continue.
```

## 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:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct node
5 {
6     int data;
7     struct node* next;
8 } node;
9
10 node* createList(node* head, int n);
11 node* sort(node* heada);
12 node* reverse(node* head);
13 node* concatinate(node* heada, node* headb);
14 void display(node* heada);
15
16 int main() {
17     int choice, n;
18     node* head = NULL;
19     node* head2 = NULL;
20     printf("Enter\n1. Create List of n nodes\n2. Sort the List\n3. Reverse the list\n");
21     printf("4. Concatenate two lists\n5. Display List\n6. Exit: ");
22     while(1) {
23         scanf("%d", &choice);
24         switch(choice) {
25             case 1:
26                 printf("Enter number of nodes to create: ");
27                 scanf("%d", &n);
28                 head = createList(head, n);
29                 break;
30             case 2:
31                 head = sort(head);
32                 break;
33             case 3:
34                 head = reverse(head);
35                 break;
36             case 4:
37                 printf("Enter number of data of list 2: ");
38                 scanf("%d", &n);
39                 head2 = createList(head2, n);
40                 head = concatinate(head, head2);
41                 break;
42             case 5:
43                 display(head);
44                 break;
45             case 6:
46                 return 0;
47             default:
48                 printf("INCORRECT ENTRY!\n");
49         }
50     }
51 }
52 }
```

```
52
53 node* createList(node* head, int n){
54     node* temp;
55     node* ptr = head;
56     int val;
57     for(int i = 0; i<n; i++){
58         temp = (node*)malloc(sizeof(node));
59         if(i==0){
60             printf("New Linked List created!\n");
61             printf("Enter the data for first node: ");
62             scanf("%d", &val);
63             temp->data = val;
64             temp->next = NULL;
65             head = temp;
66             printf("NODE 1 created!\n");
67         }
68         else{
69             ptr = head;
70             while(ptr->next!=NULL)
71                 ptr = ptr->next;
72             printf("Enter the data for node: ");
73             scanf("%d", &val);
74             temp->data = val;
75             temp->next = NULL;
76             ptr->next = temp;
77         }
78     }
79     return head;
```

```
80 }
81
82
83 node* sort(node* heada) {
84     // node* prev = heada;
85     node* templ = heada;
86     int c;
87     while(templ->next != NULL) {
88
89         node* temp2 = templ->next;
90         while (temp2 != NULL) {
91
92             if(temp2->data > templ->data) {
93                 c = templ->data;
94                 templ->data = temp2->data;
95                 temp2->data = c;
96             }
97             temp2 = temp2->next;
98         }
99         templ = templ-> next;
100    }
101    display(heada);
102    return heada;
103 }
104
105 node* reverse(node* head) {
106
107     node* prev = head;
108     node* pres = prev->next;
109     node* temp = pres;
110     prev->next = NULL;
111
112     while(temp->next!= NULL) {
113         temp = pres->next;
114         pres->next = prev;
115         prev = pres;
116         pres = temp;
117     }
```

```
118     }
119     temp->next = prev;
120     head = pres;
121
122     display(head);
123     return head;
124 }
125
126 node* concatinate(node* heada, node* headb) {
127
128     node* temp = heada;
129
130     while (temp->next != NULL) {
131         temp = temp-> next ;
132     }
133     temp->next = headb;
134     display(heada);
135     return heada;
136 }
137
138
139
140 void display(node* heada) {
141     node* temp = heada;
142     while (temp != NULL) {
143         printf("hd => ", temp->data);
144         temp = temp->next;
145     }
146     printf("NULL\n");
147 }
```

## OUTPUT:

```
Enter
1. Create List of n nodes
2. Sort the List
3. Reverse the List
4. Concatenate two lists
5. Display List
6. Exit: 1
Enter number of nodes to create: 5
New Linked List created!
Enter the data for first node: 8
NODE 1 created
Enter the data for node: 4
Enter the data for node: 6
Enter the data for node: 1
Enter the data for node: 3
5
8 -> 4 -> 6 -> 1 -> 3 -> NULL
2
8 -> 6 -> 4 -> 3 -> 1 -> NULL
3
1 -> 3 -> 4 -> 6 -> 8 -> NULL
4
Enter number of data of list 2: 6
New Linked List created!
Enter the data for first node: 1
NODE 1 created
Enter the data for node: 2
Enter the data for node: 3
Enter the data for node: 4
Enter the data for node: 5
Enter the data for node: 6
1 -> 3 -> 4 -> 6 -> 8 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> NULL
6

Process returned 0 (0x0)  execution time : 31.678 s
Press any key to continue.
```

## LAB PROGRAM 8:

WAP to implement Stack & Queues using Linked Representation

### CODE:

```
1 #include<stdio.h>
2
3 typedef struct nd{
4     int data;
5     struct nd* next;
6 }node;
7
8 node* enterElementStack(node* head, int n);
9 node* popElement(node* head);
10 void displayStack(node* head);
11 node* enterElementQueue(node* head, int n);
12 node* removeElement(node* head);
13 void displayQueue(node* head);
14
15
16 int main(){
17     char c;
18     printf("Enter you want stack(s) or queue(q): ");
19     scanf("%c", &c);
20     if(c == 'q')
21         queue();
22     else if(c == 's')
23         stack();
24     else{
25         printf("INCORRECT CHOICE!\n");
26         return 0;
27     }
28 }
29
30
31 void queue(){
32     node* head = NULL;
33     int choice;
34     int n;
35     printf("Enter choice\\n1. Enter Element\\n2. Remove Element\\n3. Disp
36     do{
37         scanf("%d", &choice);
38         switch(choice){
39             case 1:
40                 printf("Enter element: ");
41                 scanf("%d", &n);
42                 head = enterElementQueue(head, n);
43                 break;
44             case 2:
45                 head = removeElement(head);
46                 break;
47             case 3:
48                 displayQueue(head);
49                 break;
50             default:
51                 printf("Thanks! Try again!");
52         }
53     }while(choice!=4);
54     return 0;
55 }
```

```
5 /
58 void stack(){
59     node* head = NULL;
60     int choice;
61     int n;
62     printf("Enter choice\n1. Enter Element\n2. Pop Element\n");
63     do{
64         scanf("%d", &choice);
65         switch(choice){
66             case 1:
67                 printf("Enter element to enter: ");
68                 scanf("%d", &n);
69                 head = enterElementStack(head, n);
70                 break;
71             case 2:
72                 head = popElement(head);
73                 break;
74             case 3:
75                 displayStack(head);
76                 break;
77             default:
78                 printf("Thank you, please Try Again!");
79         }
    }
```

```
80     }while(choice!=4);
81 }
82
83 node* enterElementStack(node* head, int n) {
84     node* ptr;
85     node* temp;
86     if(head == NULL) {
87         printf("stack initialised\n");
88         ptr = (node*)malloc(sizeof(node));
89         head = ptr;
90         head->data = n;
91         head->next = NULL;
92     }
93     else{
94         ptr = head;
95         while(ptr->next!=NULL)
96             ptr = ptr->next;
97         temp = (node*)malloc(sizeof(node));
98         ptr->next = temp;
99         temp->data = n;
100        temp->next = NULL;
101    }
102    return head;
103 }
104
105 node* popElement(node* head) {
106     node* temp1 = head;
107     node* temp2 = head;
108     if(head==NULL){
109         printf("UNDERFLOW\n");
110         return head;
111     }
112     else{
113         while(temp1->next!=NULL)
114             temp1 = temp1->next;
115         while(temp2->next!=temp1 && temp2!=temp1)
116             temp2 = temp2->next;
117         if(head->next == NULL)
118             head = NULL;
119         temp2->next = NULL;
120         printf("Element popped is %d\n", temp1->data);
121         free(temp1);
122         return head;
123     }
124 }
125 }
```

```
126 void displayStack(node* head) {
127     node* temp = head;
128     if(head == NULL)
129         printf("UNDERFLOW\\n");
130     else{
131         while(temp!=NULL){
132             printf("%d ", temp->data);
133             temp = temp->next;
134         }
135         printf("\\n");
136     }
137 }
138
139 node* enterElementQueue(node* head, int n) {
140     node* temp = head;
141     node* ptr;
142     if(head == NULL){
143         printf("QUEUE INITIALISED\\n");
144         ptr = (node*)malloc(sizeof(node));
145         ptr->data = n;
146         ptr->next = NULL;
147         head = ptr;
148     }
149     else{
150         while(temp->next!=NULL)
151             temp = temp->next;
152         ptr = (node*)malloc(sizeof(node));
153         ptr->data = n;
154         ptr->next = NULL;
155         temp->next = ptr;
156     }
157     return head;
158 }
```

```
159
160     node* removeElement(node* head) {
161         node* temp1 = head;
162         node* ptr;
163         if(head == NULL) {
164             printf("UNDERFLOW\n");
165             return head;
166         }
167         else{
168             printf("Remove element is %d\n", head->data);
169             head = head->next;
170             free(temp1);
171         }
172         return head;
173     }
174
175     void displayQueue(node* head) {
176         node* temp = head;
177         if(head == NULL) {
178             printf("UNDERFLOW\n");
179         }
180         else{
181             while(temp!=NULL){
182                 printf("%d ", temp->data);
183                 temp = temp->next;
184             }
185             printf("\n");
186         }
187     }
188 }
```

**OUTPUT:**

```
Enter you want stack(s) of queue(q): q
Enter choice
1. Enter Element
2. Remove Element
3. Display Queue
4. Exit
1
Enter element: 1
QUEUE INITIALISED
1
Enter element: 2
1
Enter element: 3
1
Enter element: 4
1
Enter element: 5
3
1 2 3 4 5
2
Remove element is 1
2
Remove element is 2
2
Remove element is 3
2
Remove element is 4
2
Remove element is 5
2
UNDERFLOW
3
UNDERFLOW
1
Enter element: 2
QUEUE INITIALISED
2
Remove element is 2
4
Thanks! Try again!
Process returned 0 (0x0)   execution time
```

```
Enter you want stack(s) of queue(q): s
Enter choice
1. Enter Element
2. Pop Element
3. Display Stack
4. Exit
1
Enter element to enter: 1
Stack Initialised
1
Enter element to enter: 2
1
Enter element to enter: 3
3
1 2 3
2
Element popped is 3
2
Element popped is 2
2
Element popped is 1
2
UNDERFLOW
3
UNDERFLOW
4
Thank you, please Try Again!
Process returned 0 (0x0)  execution time
```

## 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:

```
1 #include<stdio.h>
2
3 typedef struct nd{
4     int data;
5     struct nd* next;
6     struct nd* prev;
7 }node;
8
9 node* createList(node* head, int n);
10 node* insertLeft(node* head, int n);
11 node* deleteElement(node* head, int n);
12 void displayList(node* head);
13
14
15 int main() {
16     int choice, n;
17     node* head = NULL;
18
19     printf("Enter 1. Creating n nodes of double linked list\n");
20     printf("2. Insert new node left of\n1. Delete specific node\n3. Display\n");
21     while(1){
22         scanf("%d", &choice);
23         switch(choice) {
24             case 1:
25                 printf("Enter number of nodes: ");
26                 scanf("%d", &n);
27                 head = createList(head, n);
28                 break;
29             case 2:
30                 printf("Enter node left to which insert is to be done: ");
31                 scanf("%d", &n);
32                 head = insertLeft(head, n);
33                 break;
34             case 3:
35                 printf("Enter node to delete: ");
36                 scanf("%d", &n);
37                 head = deleteElement(head, n);
38                 break;
39             case 4:
40                 displayList(head);
41                 break;
42             case 5:
43                 return 0;
44             default:
45                 printf("INCORRECT VALUE!\n");
46                 break;
47         }
48     }
49 }
```

```
51 node* createList(node* head, int n){  
52     node* temp;  
53     node* ptr;  
54     int i, val;  
55     for(i = 0; i<n; i++){  
56         if(i==0){  
57             printf("NEW LIST CREATED!\n");  
58             printf("Enter element: ");  
59             scanf("%d", &val);  
60             temp = (node*)malloc(sizeof(node));  
61             temp->data = val;  
62             temp->next = NULL;  
63             temp->prev = NULL;  
64             head = temp;  
65         }  
66         else{  
67             ptr = head;  
68             printf("Enter element: ");  
69             scanf("%d", &val);  
70             while(ptr->next!=NULL)  
71                 ptr = ptr->next;  
72             temp = (node*)malloc(sizeof(node));  
73             temp->data = val;  
74             temp->next = NULL;  
75             temp->prev = ptr;  
76             ptr->next = temp;  
77         }  
}
```

```
82 node* insertLeft(node* head, int n) {
83     node* ptr;
84     node* temp;
85     int val;
86     if(head == NULL){
87         printf("Value doesn't exist!\n");
88         return head;
89     }
90     else if(head->next == NULL && head->data == n){
91         printf("Enter value to insert: ");
92         scanf("%d", &val);
93         temp = (node*)malloc(sizeof(node));
94         temp->data = val;
95         temp->next = head;
96         temp->prev = NULL;
97         head->prev = temp;
98         head = temp;
99         return head;
100    }
101    else{
102        ptr = head;
103        while(ptr->data!=n && ptr->next!=NULL)
104            ptr = ptr->next;
105        if(ptr->data == n){
106            temp = (node*)malloc(sizeof(node));
107            printf("Enter value to insert: ");
108            scanf("%d", &val);
109            if(ptr->prev == NULL){
110                temp->next = head;
111                temp->data = val;
112                temp->prev = NULL;
113                head->prev = temp;
114                head = temp;
115            }
116            else{
117                temp->data = val;
118                temp->next = ptr;
119                temp->prev = ptr->prev;
120                ptr->prev->next = temp;
121                ptr->prev = temp;
122            }
123        }
124    }
125    else{
126        printf("Data doesn't exist!\n");
127    }
128 }
129 return head;
130 }
```

```
133 node* deleteElement(node* head, int n) {
134     node* temp;
135     if(head == NULL)
136         printf("EMPTY LIST!\n");
137     else{
138         temp = head;
139         while(temp->next != NULL && temp->data != n)
140             temp = temp->next;
141         if(temp->data == n){
142             if(temp->next == NULL && temp->prev == NULL){
143                 head = NULL;
144                 free(temp);
145             }
146             else if(temp->next == NULL){
147                 temp->prev->next = NULL;
148                 free(temp);
149             }
150             else if(temp->prev == NULL){
151                 temp->next->prev = NULL;
152                 head = temp->next;
153                 free(temp);
154             }
155             else{
156                 temp->next->prev = temp->prev;
157                 temp->prev->next = temp->next;
158                 free(temp);
159             }
160         }
161     }
162 }
```

```
159
160
161
162     printf("DATA DOESN'T EXIST!\n");
163 }
164 return head;
165
166
167
168 void displayList(node* head) {
169     node* temp;
170     if(head == NULL)
171         printf("EMPTY LIST!\n");
172     else{
173         temp = head;
174         while(temp!=NULL){
175             printf("%d -> ", temp->data);
176             temp = temp->next;
177         }
178         printf("NULL\n");
179     }
180 }
```

**OUTPUT:**

```
Enter
1. Creating n nodes of double linked list
2. Insert new node left of
3. Delete specific node
4. Display list
5. Exit: 1
Enter number of nodes: 5
NEW LIST CREATED!
Enter element: 1
Enter element: 2
Enter element: 3
Enter element: 4
Enter element: 5
4
1 -> 2 -> 3 -> 4 -> 5 -> NULL
2
Enter node left to which insert is to be done: 1
Enter value to insert: 7
2
Enter node left to which insert is to be done: 5
Enter value to insert: 9
2
Enter node left to which insert is to be done: 4
Enter value to insert: 13
4
7 -> 1 -> 2 -> 3 -> 13 -> 4 -> 9 -> 5 -> NULL
3
Enter node to delete: 7
3
Enter node to delete: 5
3
Enter node to delete: 3
4
1 -> 2 -> 13 -> 4 -> 9 -> NULL
5

Process returned 0 (0x0)  execution time : 98.76
Press any key to continue.
```

## 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>

typedef struct node{
    int data;
    struct node* rnext;
    struct node* lnext;
}node;

node* insertElement(node* root, int val);
void displayElements(node* root);
void preorderTraversal(node* root);
void postorderTraversal(node* root);
void inorderTraversal(node* root);

int main(){
    node* root = NULL;
    int choice, val;
    printf("1.Enter element 2.Display elements of tree\n");
    printf("3.Preorder Traversal 4.Postorder traversal 5.Inorder Traversal 6. Exit: ");
    do{
        scanf("%d", &choice);
        switch(choice){
            case 1:
                printf("Enter data to input: ");
                scanf("%d", &val);
                root = insertElement(root, val);
                break;
            case 2:
                displayElements(root);
                break;
            case 3:
                preorderTraversal(root);
                break;
            case 4:
                postorderTraversal(root);
                break;
            case 5:
                inorderTraversal(root);
                break;
        }
    }while(choice!=6);
    return 0;
}
```

```

node* insertElement(node* root, int val) {
    node *ptr, *nodeptr, *parentptr;
    ptr = (node*)malloc(sizeof(node));
    ptr->data = val;
    ptr->lnext = NULL;
    ptr->rnext = NULL;
    if(root == NULL) {
        root = ptr;
    }
    else{
        nodeptr = root;
        while(nodeptr!=NULL){
            parentptr = nodeptr;
            if(val<nodeptr->data){
                nodeptr = nodeptr->lnext;
            }
            else{
                nodeptr = nodeptr->rnext;
            }
            if(parentptr->data > val)
                parentptr->lnext = ptr;
            else
                parentptr->rnext = ptr;
        }
        return root;
    }
}

void displayElements(node* root){
    preorderTraversal(root);
}

void preorderTraversal(node* root){           //root first, then left subtree & then right
subtree
    if(root!=NULL){
        printf("%d ", root->data);

        preorderTraversal(root->lnext);
        preorderTraversal(root->rnext);
    }
}

void postorderTraversal(node* root){          //first left subtree, then right and then root
    if(root!=NULL){
        postorderTraversal(root->lnext);
        postorderTraversal(root->rnext);
        printf("%d ", root->data);
    }
}

void inorderTraversal(node* root){            //ascending order
    if(root!=NULL){
        inorderTraversal(root->lnext);
        printf("%d ", root->data);
        inorderTraversal(root->rnext);
    }
}

```

## OUTPUT:

```
Enter
1.Enter element 2.Display elements of tree
3.Preorder Traversal 4.Postorder traversal 5.Inorder Traversal 6. Exit: 1
Enter data to input: 5
1
Enter data to input: 3
1
Enter data to input: 6
1
Enter data to input: 87
1
Enter data to input: 45
1
Enter data to input: 4
2
5 3 4 6 87 45 3
5 3 4 6 87 45 4
4 3 45 87 6 5 5
3 4 5 6 45 87 6

Process returned 0 (0x0)    execution time : 61.086 s
Press any key to continue.
```