

VISVESVARAYATECHNOLOGICALUNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT On

DATA STRUCTURES (23CS3PCDST)

Submitted by

VANSH VERMA(1BM22CS318)

**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**

Dec 2023- March 2024

**B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering**



This is to certify that the Lab work entitled “**DATA STRUCTURES**” carried out by Vansh Verma(**1BM22CS318**), who is a 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 2023-24. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab - (**23CS3PCDST**) work prescribed for the said degree.

Prof. Lakshmi Neelima
Assistant Professor
Department of CSE
BMSCE, Bengaluru

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

Index Sheet

Sl. No.	Experiment Title	Page No.
1	Stack Program	4-6
2	Infix Postfix Expression	7-10
3	Queue Program	11-15
4	Circular Queue	16-18
5	Singly Linked List	19-21

6	Deletion SLL	22-27
7	Sorting SLL,Reverse SLL,Concatenation SLL	28-32
8	Stack Implementation SLL	33-35
9	Queue Implementation SLL	36-39
10	Doubly Linked List	40-43
11	Binary Search Tree	44-46
12	BFS and DFS	47-49
13	Balanced Parentheses(LeetCode)	50
14	Deletion of Middle Node(LeetCode)	51-52
15	Odd Even Linked List(LeetCode)	53-54
16	Delete a node in BST(LeetCode)	55-56
17	Binary Left Tree Value(LeetCode)	57

Course outcomes:

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.

LABPROGRAM1

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 5
void push(int); void pop();
void display(); int
stack[size],top=-1;
void main(){
    int op,n;
```

```

printf("enter the operation\n 1.push \n 2.pop\n 3.display\n enter -1 to
stop\n"); while(1){ scanf("%d",&op);
if (op== -1){ printf("stopping the
operations\n");
break;
}
else{
switch(op){
case 1:printf("enter the values\n");

scanf("%d",&n);
push(n); break;
case 2:pop();
break; case
3:display();
break;
default: printf("wrong choice\n");
}
}
}
}
void push(int n){ if(top==size-
1){
printf("stack overflow condition\n");
}
else{
top++;
stack[top]=n;
printf("push operation is succesfull\n");
}
}
void pop(){
if(top== -1){
printf("stack underflow condition\n");

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

```

```

        printf("%d\t", stack[i]);
    }
    printf("\n");
}
}

```

OUTPUT

```

enter the operation
1.push
2.pop
3.display
enter -1 to stop
1
enter the values
10
push operation is succesfull
1
enter the values
20
push operation is succesfull
2
20 pop() operation successfull
3
10
-1
stopping the operations

Process returned 0 (0x0)    execution time : 23.158 s
Press any key to continue.

```

LABPROGRAM2

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

char stack[100];
int top = -1,size;

void push(char item)
{
    if(top >= size-1)
    {
        printf("\nStack Overflow.");
    }
    else
    {
        top = top+1;
        stack[top] = item;
    }
}

char pop()
{
    char item ;
    if(top <0)
    {
        printf("\nStack Underflow\n");
    }
    else
    {
        item = stack[top];
        top = top-1;
        return(item);
    }
}

int is_operator(char symbol)
{
    if(symbol == '^' || symbol == '*' || symbol == '/' || symbol == '+' || symbol == '-')
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

```

```

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

void InfixToPostfix(char infix_exp[], char postfix_exp[])
{ int i, j;
  char item;
  char x;

  push('(');
  strcat(infix_exp, "(");

  i=0; j=0;
  item=infix_exp[i]
  ; while(item !=
  '\0')
  {
      if(item == '(')
      {
          push(item);
      }
      else if( isdigit(item) || isalpha(item))
      {
          postfix_exp[j] = item;
          j++;
      }
      else if(is_operator(item) == 1)
      {
          x=pop(); while(is_operator(x) == 1 && precedence(x)>=
          precedence(item))

```

```

        {
            postfix_exp[j] = x;
            j++;
            x = pop();
        }
        push(x);

        push(item);
    }
    else if(item == ')')
    {
        x = pop();
        while(x != '(')
        {
            postfix_exp[j] = x;
            j++;
            x = pop();
        }
    }
    else
    { printf("\nInvalid infix Expression.\n");
      exit(1);
    } i++; item =
    infix_exp[i];
}
postfix_exp[j] = '\0';
}
main()
{
    char  infix[100], postfix[100]; printf("\nEnter size of stack"); scanf("%d",&size);
    printf("Assume the infix expression contains single letter variables and single digit
constants only.\n"); printf("\nEnter
Infix expression : "); scanf("
%s",infix);
    InfixToPostfix(infix,postfix);
    printf("Postfix Expression: ");
    printf("%s",postfix); }

```


OUTPUT

```
Enter size of stack 3
Assume the infix expression contains single letter variables and single digit constants only.

Enter Infix expression : 2a+6b+m*k-k
Postfix Expression: 2a6b+mk**k-
Process returned 0 (0x0)   execution time : 41.382 s
Press any key to continue.
```

LABPROGRAM3

write a program to simulate the working of the queue of integers using an array. Provide the following operations: Insert, delete, display. The program should print appropriate message for overflow and underflow condition

CODE

```
#include<stdio.h>
#include<conio.h>
#define MAX 3 int
queue[MAX]; int front=-
1, rear=-1; void
insert(void); int
delete_element(void); int
peek(void); void
display(void); int main()
{ int option, val;
  do
  {
    printf("\n\n****MAIN
    MENU****"); printf("\n 1.Insert
    an element"); printf("\n 2.Delete
    an element"); printf("\n 3.Peek");
    printf("\n 4.Display the queue");
    printf("\n 5.Exit"); printf("\n Enter
    your option:");
    scanf("%d",&option);
    switch(option)
    {
      case 1:
        insert();
        break;
      case 2:
```

```

        val=delete_element(); if(val!=-
        1)
            printf("\n The number deleted is :%d",val);
            break;
        case 3: val=peek(); if(val!=-1) printf("\n The first
            value in queue is:%d",val);
            break;
        case 4:
            display();
            break;
    }
}while(option!=5)
; getch(); return
0;
}
void insert()
{
    int num; printf("\n Enter the number to be inserted in the
    queue:"); scanf("%d",&num); if(rear==MAX-1)
        printf("\n OVERFLOW");
    else if(front==-1 &&rear==-1)
        front=rear=0;
    else
        rear++;
    queue[rear]=num;
}
int delete_element()
{ int val; if(front==
1||front>rear)
    {
        printf("\n UNDERFLOW");
        return -1;
    }
    else
    {
        val=queue[front];
        front++;
        if(front>rear)
            front=rear=-1;
        return val;
    }
}
int peek()

```

```

{ if(front==-1||front>rear)
  {
    printf("\n QUEUE IS EMPTY");
    return -1;
  }
  else
  {
    return queue[front];
  }
}
void display()
{ int i; printf("\n");
  if(front==-1||front>rear)
    printf("\n QUEUE IS EMPTY");
  else
  { for(i=front;i<=rear;i++)
    printf("\t %d",queue[i]);
  }
}

```

OUTPUT

```
****MAIN MENU****
1.Insert an element
2.Delete an element
3.Peek
4.Display the queue
5.Exit
Enter your option:4
```

QUEUE IS EMPTY

```
****MAIN MENU****
1.Insert an element
2.Delete an element
3.Peek
4.Display the queue
5.Exit
Enter your option:2
```

UNDERFLOW

```
****MAIN MENU****
1.Insert an element
2.Delete an element
3.Peek
4.Display the queue
5.Exit
Enter your option:1
```

Enter the number to be inserted in the queue:10

```
****MAIN MENU****
1.Insert an element
2.Delete an element
3.Peek
4.Display the queue
5.Exit
Enter your option:1
```

Enter the number to be inserted in the queue:20

```
****MAIN MENU****
1.Insert an element
2.Delete an element
3.Peek
4.Display the queue
5.Exit
Enter your option:1
```

Enter the number to be inserted in the queue:30

Enter your option:1

Enter the number to be inserted in the queue:40

OVERFLOW

****MAIN MENU****

- 1.Insert an element
- 2.Delete an element
- 3.Peek
- 4.Display the queue
- 5.Exit

Enter your option:2

The number deleted is :10

****MAIN MENU****

- 1.Insert an element
- 2.Delete an element
- 3.Peek
- 4.Display the queue
- 5.Exit

Enter your option:2

The number deleted is :20

****MAIN MENU****

- 1.Insert an element
- 2.Delete an element
- 3.Peek
- 4.Display the queue
- 5.Exit

Enter your option:3

The first value in queue is:40

****MAIN MENU****

- 1.Insert an element
- 2.Delete an element
- 3.Peek
- 4.Display the queue
- 5.Exit

Enter your option:4

40

****MAIN MENU****

- 1.Insert an element
- 2.Delete an element
- 3.Peek
- 4.Display the queue
- 5.Exit

Enter your option:5

Process returned 0 (0x0) execution time : 79.289 s

Press any key to continue.

LABPROGRAM4

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

CODE

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#define QUE_SIZE 3
int item,front=0,rear=-1,q[QUE_SIZE],count=0; void insertrear()
{
if(count==QUE_SIZE)
{
printf("queue overflow\n");
return;
}
rear=(rear+1)%QUE_SIZE;
q[rear]=item; count++;
} int
deletefront()
{

if(count==0) return -1; item=q[front]; front=(front+1)%QUE_SIZE; count=count-1;
return item;
}
void displayQ() {
int i,f;
if(count==0)
{
printf("queue is empty\n"); return;
} f=front; printf("Contents of queue \n");
for(i=1;i<=count;i++)
{
printf("%d\n",q[f]); f=(f+1)%QUE_SIZE;
}
}
void main()
{
```

```

int choice;
for(;;)
{ printf("\n1:insertrear\n2:deletefront\n3:display\n4:exit\n"); printf("enter the choice\n");
  scanf("%d",&choice); switch(choice)
  {
    case 1:printf("enter the item to be inserted\n");
      scanf("%d",&item); insertrear(); break;
    case 2:item=deletefront(); if(item==-1)
      printf("queue is empty\n"); else
      printf("item deleted =%d\n",item); break;
    case 3:displayQ(); break;
    default:exit(0); } } getch();

}

```


OUTPUT

```
1:insertrear
2:deletefront
3:display
4:exit
enter the choice
1
enter the item to be inserted
10

1:insertrear
2:deletefront
3:display
4:exit
enter the choice
1
enter the item to be inserted
20

1:insertrear
2:deletefront
3:display
4:exit
enter the choice
1
enter the item to be inserted
30

1:insertrear
2:deletefront
3:display
4:exit
enter the choice
2
item deleted =10

1:insertrear
2:deletefront
3:display
4:exit
enter the choice
3
Contents of queue
20
30

1:insertrear
2:deletefront
3:display
4:exit
enter the choice
4

Process returned 0 (0x0)   execution time : 23.786 s
Press any key to continue.
```

LABPROGRAM5

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.

Display the contents of the linked list.

CODE

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data; struct
node *next;
};
void printData(struct node *head)
{
if(head==NULL)
{
printf("The list is empty");
}else{ struct node
*ptr=head;
while(ptr!=NULL)
{ printf("%d\n",ptr-
>data); ptr=ptr->next;
}}
}
void insertBeg(struct node **head,int value)
{
struct node *temp=(struct node*)malloc(sizeof(struct node));
temp->data=value; temp->next=*head;
*head=temp;
}
void insertEnd(struct node*head, int value)
{
struct node *ptr=head; struct node *temp=(struct
node*)malloc(sizeof(struct node)); temp->data=value; temp-
>next=NULL; while(ptr->next!=NULL){ ptr=ptr->next;
}
ptr->next=temp;
}
```

```

void insertAtPos(struct node *head,int value,int pos)
{
    struct node *ptr,*ptr2; struct node *temp=(struct
    node*)malloc(sizeof(struct node)); temp->data=value; temp-
    >next=NULL; int position=pos; ptr=head; while(pos!=1)
    { ptr2=ptr;
    ptr=ptr->next;
    pos--;
    }
    temp->next=ptr2->next; ptr2->next=temp; printf("value %d
    added succuessful at %d\n",value,position);
}

int main()
{
    struct node *head=NULL;
    insertBeg(&head,34);
    printData(head);
    printf("-----\n");
    insertEnd(head,75);
    insertEnd(head,56);
    insertEnd(head,87);
    printData(head);
    printf("-----\n");
    insertAtPos(head,89,3);
    printData(head);
}

```

OUTPUT

```
34
-----
34
75
56
87
-----
value 89 added succuessful at 3
34
75
89
56
87

Process returned 0 (0x0)   execution time : 0.068 s
Press any key to continue.
|
```

LABPROGRAM6

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.

CODE

```
#include<stdio.h>

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

struct node *head=NULL,*newnode,*temp;

void create()
{ int i,n;
  printf("enter the number of elements:\n");
  scanf("%d",&n);

  for(i=0;i<n;i++)
  {
```

```

newnode=(struct node *)malloc(sizeof(struct node));
printf("Enter the element %d:\n",i+1);
scanf("%d",&newnode->data); newnode-
>next=NULL;

if(head==NULL)
{
    temp=head=newnode;
}
else{
    temp->next=newnode;
    temp=newnode;
}
}
}
void display()
{
    temp=head; printf("The
elements are:\n");
while(temp!=NULL)
{
    printf("%d\n",temp->data);
    temp=temp->next;
}
}

void delete_beg()
{
    temp=head;
    if(head==NULL)
    {
        printf("List is empty\n");
    }
    else
    {
        head=temp->next;
        free(temp);
    }
}

void delete_end()
{
    temp=head; struct
node *prenode;
while(temp->next!=NULL)
{
    prenode=temp;

```

```

        temp=temp->next;
    }
    if(temp==head)
    {
        head=NULL;
    }
    else
    {
        prenode->next=NULL;
    }
    free(temp);
}

```

```

void delete_pos()
{
    struct node *nextnode; int
    pos,i=1; printf("enter the
    position:\n");
    scanf("%d",&pos);
    temp=head; while(i<pos-1)
    {
        temp=temp->next;
        i++;
    }
    nextnode=temp->next; temp->next=nextnode-
    >next;
    free(nextnode);
}

```

```

void main()
{
    int choice;

    while(1)
    {
        printf("Enter operation:\n1.create\n2.display\n3.delete at beginnning\n4.delete at
        end\n5.delete at position\n6.-1 to end\n");
        scanf("%d",&choice);

        if(choice==1)
        {
            printf("operation completed!\n");
            break;
        }
        else
        {

```

```
switch(choice)
{
    case 1:create();
    break; case
    2:display(); break;
    case
    3:delete_beg();
    break; case
    4:delete_end();
    break; case
    5:delete_pos();
    break;
    default:printf("invalid output\n");
}
}
}
}
```

OUTPUT

```
Enter operation:
1.create
2.display
3.delete at beginnning
4.delete at end
5.delete at position
6.-1 to end

1
enter the number of elements:
4
Enter the element 1:
10
Enter the element 2:
20
Enter the element 3:
30
Enter the element 4:
40
Enter operation:
1.create
2.display
3.delete at beginnning
4.delete at end
5.delete at position
6.-1 to end

2
The elements are:
10
20
30
40
Enter operation:
1.create
2.display
3.delete at beginnning
4.delete at end
5.delete at position
6.-1 to end

3
```



```
Enter operation:
1.create
2.display
3.delete at beginnning
4.delete at end
5.delete at position
6.-1 to end
2
The elements are:
20
30
40
Enter operation:
1.create
2.display
3.delete at beginnning
4.delete at end
5.delete at position
6.-1 to end

3
Enter operation:
1.create
2.display
3.delete at beginnning
4.delete at end
5.delete at position
6.-1 to end
2
The elements are:
30
40
Enter operation:
1.create
2.display
3.delete at beginnning
4.delete at end
5.delete at position
```

```
Enter operation:
1.create
2.display
3.delete at beginnning
4.delete at end
5.delete at position
6.-1 to end
5
enter the position:
2
Enter operation:
1.create
2.display
3.delete at beginnning
4.delete at end
5.delete at position
6.-1 to end
-1
```

LABPROGRAM7

All-sort,reverse,concatenation.

CODE

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

struct node
{
    int data; struct
    node *next;

};

void append(struct node **head, int new_data)
{
    struct node *new_node = (struct node *)malloc(sizeof(struct node));

    new_node->data = new_data;
    new_node->next = NULL;
    struct node *last = *head;

    if (*head == NULL)
        *head = new_node;
    else
    {
        while (last->next != NULL)
            last = last->next;

        last->next = new_node;
    }
}

void display(struct node *head)
{
    if (head == NULL)
    {
        printf("Linked List
        empty.\n"); return; }
    printf("Linked List:");
```

```

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

```

```

void bubble_sort(struct node *head)
{
    struct node
    *prev; struct
    node *cur; int
    nex; int flag = 1;
    int flag2 = 1;

    while (flag)
    {
        prev = head; while (prev != NULL && prev-
        >next != NULL)
        {
            cur = prev->next;

            if (cur->data < prev->data)
            {
                nex = cur->data; cur-
                >data = prev->data; prev-
                >data = nex;
            }

            prev = prev->next;
        }

        int max = 0;
        prev = head;

        while (prev != NULL)
        {
            if (max > prev->data)
            {

```

```

        flag2 = 0;
        break;
    }
    max = prev->data;
    prev = prev->next;
}

if (flag2)
    flag = 0;
else
    flag2 = 1;
}
}

void reverse(struct node **head)
{
    struct node *prev = NULL;
    struct node *current = *head;
    struct node *next = NULL;

    while (current != NULL)
    {
        next = current->next; current->
        next = prev;

        prev = current;
        current = next;
    }

    *head = prev;
}

void concat(struct node *head1, struct node *head2){
    struct node *prev=head2;

    while(prev!=NULL){ append(&head1,prev->
        data);
        prev=prev->next;
    }
}

```

```

int main()
{
    struct node *head=NULL;
    int choice;
    append(&head,5);
    append(&head,2);
    append(&head,3);
    append(&head,4);
    append(&head,1);
    append(&head,6); struct
    node *head2=NULL;

    while (1)
    { printf("-----\n");
      printf("1.Bubble Sort\n2.Reverse\n3.Concat\nChoice:");
      scanf("%d",&choice);
      printf("-----\n");
      switch (choice)
      {
          case 1:bubble_sort(head);
                display(head);

                break;
          case 2: reverse(&head);
                display(head);
                break;

          case 3:
                append(&head2,76);
                append(&head2,43);
                append(&head2,34);

                concat(head,head2)
                ; display(head);
                break;

      }
    }
    return 0;
}

```

OUTPUT

```
-----  
1.Bubble Sort  
2.Reverse  
3.Concat  
Choice:1  
-----  
Linked List:1 2 3 4 5 6  
-----  
1.Bubble Sort  
2.Reverse  
3.Concat  
Choice:2  
-----  
Linked List:6 5 4 3 2 1  
-----  
1.Bubble Sort  
2.Reverse  
3.Concat  
Choice:3  
-----  
Linked List:6 5 4 3 2 1 76 43 34  
-----  
1.Bubble Sort  
2.Reverse  
3.Concat  
Choice:|
```

LABPROGRAM8

Stack implementation using single linked list

CODE

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

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

```
void append(struct node** head, int new_data) {
```

```

struct node* new_node = (struct node*)malloc(sizeof(struct node));

new_node->data = new_data;
new_node->next = NULL;
struct node* last = *head;

if (*head == NULL)
    *head = new_node;
else {
    while (last->next != NULL)
        last = last->next;

    last->next = new_node;
}
}

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

void del_end(struct node *head){
    if(head==NULL){
        printf("List Empty\n");
        return;
    }
    struct node *last=head;

    struct node *prev; while(last-
>next!=NULL){
        prev=last; last=last-
>next;
    }
    free(last);
    prev->next=NULL;
}

```



```
}
```

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

```
int choice,a;
```

```
while(choice<4){  
    printf("1.Push\n2.Pop\n3.Display\nChoice:")  
    ; scanf("%d",&choice); switch (choice)  
    {  
    case 1:  
        printf("Enter  
        value:");  
        scanf("%d",&a);  
        append(&head,a);  
        display(head);  
        break;  
    case 2:  
        del_end(head);  
        display(head);  
        break;  
    case 3:  
        display(head);  
    default:  
        break;  
    }  
}
```

```
return 0;
```

```
}
```

OUTPUT

```
1.Push
2.Pop
3.Display
Choice:1
Enter value:10
Stack:10
1.Push
2.Pop
3.Display
Choice:1
Enter value:20
Stack:10 20
1.Push
2.Pop
3.Display
Choice:2
Stack:10
1.Push
2.Pop
3.Display
Choice:3
Stack:10
1.Push
2.Pop
3.Display
Choice:|
```

LABPROGRAM8

Queue implementation using single linked list

CODE

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

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

```

struct Queue {
    struct Node* front;
    struct Node* rear;
};

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(EXIT_FAILURE);
    }
    newNode->data = data; newNode->next = NULL;
    return newNode;
}

struct Queue* initializeQueue() {
    struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
    if (queue == NULL) {
        printf("Memory allocation failed.\n");
        exit(EXIT_FAILURE);
    }
    queue->front = queue->rear = NULL;
    return queue;
}

void enqueue(struct Queue* queue, int data) {
    struct Node* newNode = createNode(data);

    if (queue->rear == NULL) {
        queue->front = queue->rear = newNode;
        return;
    }

    queue->rear->next = newNode;
    queue->rear = newNode;
}

void dequeue(struct Queue* queue) {
    if (queue->front == NULL) { printf("Queue underflow. Cannot dequeue.\n"); return; }
}

struct Node* temp = queue->front;
queue->front = queue->front->next;

if (queue->front == NULL) {
    queue->rear = NULL;
}

```

```
free(temp);  
}
```

```
void displayQueue(struct Queue* queue) {  
    if (queue->front == NULL) { printf("Queue  
is empty.\n"); return;  
}
```

```
    struct Node* current = queue->front;  
    printf("Queue: "); while  
    (current != NULL) {  
        printf("%d ", current->data);  
        current = current->next;  
    }  
    printf("\n");  
}
```

```
void freeQueue(struct Queue* queue) {  
    while (queue->front != NULL) { struct  
    Node* temp = queue->front; queue-  
    >front = queue->front->next;  
    free(temp);  
    }  
    free(queue);  
}
```

```
int main() { struct Queue* queue =  
initializeQueue(); int choice, data;
```

```
do {  
    printf("\nMenu:\n");  
    printf("1. Enqueue\n");  
    printf("2. Dequeue\n");  
    printf("3. Display\n");  
    printf("4. Exit\n");
```

```
    printf("Enter your choice: ");  
    scanf("%d", &choice); switch (choice) {  
    case 1:  
        printf("Enter data to enqueue: ");  
        scanf("%d", &data);  
        enqueue(queue, data); break;
```

```
    case 2:  
        dequeue(queue);  
        break;
```

```
    case 3:  
        displayQueue(queue);  
        break;
```

```
case 4:
printf("Exiting the program.\n");
break;

default: printf("Invalid choice! Please enter a valid
option.\n");
}

} while (choice != 4);

freeQueue(queue);

return 0;
}
```

OUTPUT

```
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter data to enqueue: 10
```

```
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter data to enqueue: 20
```

```
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter data to enqueue: 30
```

```
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
```

```
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue: 20 30
```

```
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4
Exiting the program.
```

```
Process returned 0 (0x0)   execution time : 20.029 s
Press any key to continue.
```

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

CODE

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

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

node* head = NULL;
int count = 0;

void insert(int data, int position);
void delete(int element); void
display();

int main(){
    int data, choice, pos;
    printf("1. Insert\n2. Delete\n3. Exit\nChoice: ");
    scanf("%d", &choice);
    while(choice != 3){
        if (choice == 1){
            printf("Enter data and position:
"); scanf("%d%d", &data, &pos);
            insert(data, pos); printf("Count:
%d\n", count);
        } else if (choice == 2){
            printf("Enter element: ");
            scanf("%d", &pos);
            delete(pos); printf("Count:
%d\n", count);
        } display();
        printf("Enter choice:
");
        scanf("%d", &choice);
    }

    return 0;
```



```

}

void insert(int data, int position){
    if (position == 0){
        node* new_node = malloc(sizeof(node));
        new_node->data = data; new_node->next
        = head; new_node->prev = NULL; if (head
        != NULL) head->prev = new_node; head =
        new_node; count++;
        return;
    } else if (position == count){ node*
        new_node = malloc(sizeof(node));
        new_node->data = data; new_node-
        >next = NULL; node* temp = head;
        while(temp->next != NULL) temp =
        temp->next;
        temp->next = new_node;
        new_node->prev =
        temp; count++; return;

    } else if (position > count || position < 0){
        printf("Unable to insert at given position\n");
        return;
    } else { node* temp =
        head;
        for(int i = 0; i < position-1; i++)
            temp = temp->next;
        node* new_node =
        malloc(sizeof(node)); new_node->data
        = data; new_node->next = temp->next;
        new_node->prev = temp; temp->next-
        >prev = new_node; temp->next =
        new_node; count++;
        return;
    }
}

void delete(int element){ int position =
    0; node *temp = head; if (head ==
    NULL){
        printf("List is empty, cannot delete"); return;
    }
    for(; position < count; temp=temp->next, position++)
        if (temp->data == element) break;
    if (temp == NULL){
        printf("Element does not exist in list"); return;
    }
}

```

```

if (position == 0){
    node* temp = head;
    temp = temp->next;
    temp->prev =
    NULL;
    free(head);
    head = temp; count-
    -;
    return;
} else if (position == count-1){
    node* temp = head; for(int i
    = 1; i < count-1; i++) temp
    = temp->next;
    node* temp1 = temp->next;
    temp->next =
    NULL; free(temp1);
    count--;
    return;
} else if (position > count || position < 0){
    printf("Unable to delete at position\n");
    return;
} else { node* temp =
    head;
    for(int i = 0; i < position; i++)
        temp = temp->next;
    temp->next->prev = temp-
    >prev; temp->prev->next =
    temp->next; free(temp); count--;
    return;
}
}

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

```

OUTPUT

```
1. Insert
2. Delete
3. Exit
Choice: 1
Enter data and position: 10 0
Count: 1
Linked List: 10
Enter choice: 1
Enter data and position: 20 1
Count: 2
Linked List: 10 20
Enter choice: 2
Enter element: 10
Count: 1
Linked List: 20
Enter choice:
```

LABPROGRAM10

Tree Program(inorder, postorder, preorder)

CODE

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

```
typedef struct Node{
int data; struct Node
*left; struct Node
*right; } node; node
*root = NULL;
```

```
void insert(node **root, int data);
void preorder(node **root); void
postorder(node **root); void
inorder(node **root);
```

```

int main(){
    int choice, data; insert(&root, 8); insert(&root, 3); insert(&root,
    1); insert(&root, 6); insert(&root, 4); insert(&root, 7);
    insert(&root, 10); insert(&root, 14); insert(&root, 13); printf("1.
    Preorder\n2. Inorder\n3. Postorder\n4. Exit\nChoice: ");
    scanf("%d", &choice); while (choice != 4){ if (choice == 1){
    preorder(&root); printf("\n");
        } else if (choice == 2){
            inorder(&root);
            printf("\n");
        } else if (choice == 3){
            postorder(&root);
            printf("\n");
        }
        printf("Enter choice: ");
        scanf("%d", &choice);
    }
}

```

```

void insert(node **root, int data){
    if (*root == NULL) {
        node *new_node = malloc(sizeof(node));
        new_node->data = data; new_node->
        right = NULL; new_node->left = NULL;
        *root = new_node;
        return;
    }
    if (data < (*root)->data){
        insert(&((*root)->left), data);
    } else if (data > (*root)->data){ insert(&((*root)-
        >right), data);
    }
    return;
}

```

```

void preorder(node **root){
    if (*root != NULL){
        printf("%d ", (*root)->data); preorder(&((*root)-
        >left)); preorder(&((*root)->right));
    }
}

```

```

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

```

```

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

```

OUTPUT

```

1. Preorder
2. Inorder
3. Postorder
4. Exit
Choice: 1
8 3 1 6 4 7 10 14 13
Enter choice: 2
1 3 4 6 7 8 10 13 14
Enter choice: 3
1 4 7 6 3 13 14 10 8
Enter choice: 4

Process returned 0 (0x0)   execution time : 14.294 s
Press any key to continue.
|

```

LABPROGRAM11

BFS & DFS

CODE

```

#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100
struct Node { int data;
struct Node* next;

```

```

};
struct Graph { int
numVertices; struct
Node** adjLists; int*
visited;
};
struct Node* createNode(int data) {
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->data = data; newNode->next = NULL; return newNode;
}
struct Graph* createGraph(int numVertices) { struct Graph* graph = (struct
Graph*)malloc(sizeof(struct Graph)); graph->numVertices = numVertices;
graph->adjLists = (struct Node*)malloc(numVertices * sizeof(struct
Node)); graph->visited = (int*)malloc(numVertices * sizeof(int)); for (int i =
0; i < numVertices; i++) { graph->adjLists[i] = NULL; graph->visited[i] = 0;
}
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 BFS(struct Graph* graph, int startVertex) { int
queue[MAX_SIZE]; int front = -1, rear = -1; graph-
>visited[startVertex] = 1; queue[++rear] =
startVertex; while (front != rear) { int currentVertex =
queue[++front]; printf("%d ", currentVertex); struct
Node* temp = graph->adjLists[currentVertex]; while
(temp) { int adjVertex = temp->data; if(graph-
>visited[adjVertex] == 0) { graph->visited[adjVertex]
= 1; queue[++rear] = adjVertex;
}
temp = temp->next;
}
}
}
}

```

```

void DFS(struct Graph* graph, int vertex) {
graph->visited[vertex] = 1; printf("%d ",
vertex); struct Node* temp = graph-
>adjLists[vertex]; while (temp) { int
adjVertex = temp->data; if (graph-
>visited[adjVertex] == 0) {
DFS(graph, adjVertex);
}
temp = temp->next;
}
}
int main(){ struct Graph* graph =
createGraph(4); addEdge(graph, 0, 1);
addEdge(graph, 0, 2); addEdge(graph, 1,
2); addEdge(graph, 2, 3); printf("BFS\n");
BFS(graph, 0); for (int i = 0; i < graph-
>numVertices; i++){ graph->visited[i] = 0;
}
printf("\nDFS\n");
DFS(graph, 0);
return 0;
}

```

OUTPUT

```

BFS
0 2 1 3
DFS
0 2 3 1
Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.

```

LABPROGRAM12

Balanced Parentheses(LeetCode)

CODE

```
#include <stdio.h>
#include <string.h>
int scoreOfParentheses(char *s)
{
    int stack[50];
    int top = -1;
    int score = 0;
    for (int i = 0; s[i] != '\0'; i++)
    { if (s[i] == '(')
        {
            stack[++top] = score;
            score = 0;
        }
        else
        {
            score = stack[top--] + (score == 0 ? 1 : 2 * score);
        }
    }
    return score;
}
```

OUTPUT

Accepted Runtime: 0 ms

• Case 1

• Case 2

• Case 3

Input

s =
"()"

Output

1

Expected

1

LABPROGRAM13

Delete the Middle Node of a Linked List(LeetCode)

CODE

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

struct Node* deleteMiddle(struct Node* head) {
    if (head == NULL)
        return NULL;
    if (head->next == NULL)
        { free(head); return
        NULL;
    }

    struct Node* slow_ptr = head;
    struct Node* fast_ptr = head;
    struct Node* prev;
    while (fast_ptr != NULL && fast_ptr->next != NULL) {
        fast_ptr = fast_ptr->next-
        >next; prev = slow_ptr;
        slow_ptr = slow_ptr->next;
    }

    prev->next = slow_ptr->next;
    free(slow_ptr);

    return head;
}

void printList(struct Node* head) {
    struct Node* current = head;
    while (current != NULL) {
```

```

printf("%d -> ", current->data);
current = current->next;
}
printf("NULL\n");
}

```

OUTPUT

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

head =
[1,3,4,7,1,2,6]

Output

[1,3,4,1,2,6]

Expected

[1,3,4,1,2,6]

LABPROGRAM14

Odd Even Linked List

CODE

```

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

struct ListNode {
    int val;
    struct ListNode *next;
};

struct ListNode* oddEvenList(struct ListNode* head) { if (head == NULL
|| head->next == NULL || head->next->next == NULL) return head;

    struct ListNode *odd = head; struct
    ListNode *even = head->next; struct
    ListNode *evenHead = even;

    while (even != NULL && even->next != NULL) {

```

```
odd->next = even->next;
odd = odd->next; even-
>next = odd->next;
even = even->next;
}

odd->next = evenHead;
return head;
}
struct ListNode* newNode(int val)
{
struct ListNode* node = (struct ListNode*)malloc(sizeof(struct ListNode));
node->val = val; node->next = NULL;
return node;
}
```

OUTPUT

Accepted Runtime: 0 ms

• Case 1

• Case 2

Input

head =
[1,2,3,4,5]

Output

[1,3,5,2,4]

Expected

[1,3,5,2,4]

LABPROGRAM15

Delete a node in BST .

CODE

```
struct TreeNode* minValueNode(struct TreeNode* node)
{
    struct TreeNode* current = node; while
    (current && current->left != NULL)
    current = current->left;
    return current;
}
struct TreeNode* deleteNode(struct TreeNode* root, int key) {
    if (root == NULL) return root; if (key < root->val) root->left =
    deleteNode(root->left, key); else if (key > root->val) root-
    >right = deleteNode(root->right, key); else { if (root->left ==
    NULL) {
        struct TreeNode* temp = root->right;
        free(root); return temp;
    } else if (root->right == NULL) {
        struct TreeNode* temp = root->left;
        free(root); return temp;
    }
    struct TreeNode* temp = minValueNode(root->right);
    root->val = temp->val;
    root->right = deleteNode(root->right, temp->val);
}
return root;
}
```

OUTPUT

Accepted Runtime: 3 ms

- Case 1
- Case 2
- Case 3

Input

root =
[5,3,6,2,4,null,7]

key =
3

Output

[5,4,6,2,null,null,7]

LABPROGRAM16

Bottom Left Tree Value.

CODE

```
void findBottomLeft(struct TreeNode* node, int depth, int* maxDepth, int* leftmostValue) {
    if (node == NULL)
        return;

    if (depth > *maxDepth) {
        *maxDepth = depth;
        *leftmostValue = node->val;
    }


    findBottomLeft(node->left, depth + 1, maxDepth, leftmostValue);
    findBottomLeft(node->right, depth + 1, maxDepth, leftmostValue);
}

int findBottomLeftValue(struct TreeNode* root) {
    int maxDepth = 0; int leftmostValue = root->val;

    findBottomLeft(root, 1, &maxDepth, &leftmostValue);

    return leftmostValue;
}
```

OUTPUT

Accepted Runtime: 3 ms 

• Case 1

• Case 2

Input

root =
[2,1,3]

Output

1

Expected

1