

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT On

DATA STRUCTURES (23CS3PCDST)

Submitted by

VARNITH D RAMESH (1BM22CS319)

**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**

Submittted to

Prof. Surabhi S
Assistant Professor
Department of CSE
BMSCE, Bengaluru



**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**” was carried out by **VARNITH D RAMESH (1BM22CS319)**, 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 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. Surabhi S
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 | Program 1: Write a program to implement basic stack operations such as PUSH, POP, and Display. | 5 |
| 2 | Program 2: Write a program to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide). | 7 |
| 3 | Program 3a: Write a program to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display The program should print appropriate messages for queue empty and queue overflow conditions. Program 3b: WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & Display The program should print appropriate messages for queue empty and queue overflow conditions. | 9 |
| 4 | Program 4: Write a program to implement Singly Linked List with the following operations <ul style="list-style-type: none"> a. Create a linked list. b. Insertion of a node at the first position, at any position, and at the end of the list. c. deletion of the first element, specified element, and last element in the list. d. Display the contents of the linked list. | 15 |
| 5 | Program 5a: Write a program to implement a Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists Program 5b: Write a program to implement a Single Link List to simulate Stack & Queue Operations. | 18 |
| 6 | Program 6: Write a program to Implement doubly link list with primitive operations <ul style="list-style-type: none"> 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 Leetcode-856 | 26 |
| 7 | Program 7: Write a program | 31 |

| | | |
|---|---|----|
| | a. To construct a binary Search tree. b. To traverse the tree using all the methods i.e., in-order, pre order and post order to display the elements in the tree. Leetcode-876 Leetcode-328 | |
| 8 | Program 8a: Write a program to traverse a graph using the BFS method. Program 8b: Write a program to check whether a graph is connected or not using the DFS method. Leetcode-513 Leetcode-450 | 38 |

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. |

Lab Program 1:

Write a program to implement basic stack operations such as PUSH, POP, and Display.

Code:

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

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

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

char pop()
{
    if(top == -1)
        return -1;
    else
        return stack[top--];
}

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

int main()
{
    char exp[100];
    char *e, x;
    printf("Enter the expression : ");
    scanf("%s",exp);
    printf("\n");
    e = exp;

    while(*e != '\0')
    {
        if(isalnum(*e))
            printf("%c ",*e);
        else if(*e == '(')
```

```

        push(*e);
    else if(*e == ')')
    {
        while((x = pop()) != '(')
            printf("%c ", x);
    }
    else
    {
        while(priority(stack[top]) >= priority(*e))
            printf("%c ",pop());
        push(*e);
    }
    e++;
}

while(top != -1)
{
    printf("%c ",pop());
}return 0;
}

```

Output:

```

1.PUSH 2.POP 3.DISPLAY 4.EXIT
3
Stack underflow
1.PUSH 2.POP 3.DISPLAY 4.EXIT
1
Enter a value:23
1.PUSH 2.POP 3.DISPLAY 4.EXIT
1
Enter a value:45
1.PUSH 2.POP 3.DISPLAY 4.EXIT
1
Enter a value:67
1.PUSH 2.POP 3.DISPLAY 4.EXIT
1
Enter a value:123
1.PUSH 2.POP 3.DISPLAY 4.EXIT
1
Enter a value:23
1.PUSH 2.POP 3.DISPLAY 4.EXIT
1
Enter a value:423
Stack overflow
1.PUSH 2.POP 3.DISPLAY 4.EXIT
3
23      45      67      123    23
1.PUSH 2.POP 3.DISPLAY 4.EXIT
2
23 has been deleted from the Stack
1.PUSH 2.POP 3.DISPLAY 4.EXIT
4

```

Lab Program 2:

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

Code:

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5
int arr[5], top=-1;
void push(int value){
    if(top==SIZE-1){
        printf("\nStack overflow");
    }else{

        top+=1;
        arr[top]=value;
    }
}
void pop(){
    if(top== -1){
        printf("\nStack underflow");
    }else{
        printf("\n%d has been deleted from the Stack",arr[top]);
        top=top-1;
    }
}
void display(){
    if(top== -1){
        printf("\nStack underflow ");
    }else{
        for(int i=0;i<=top;i++){
            printf("%d\t",arr[i]);
        }
    }
}
int main(){
    int choice,value;
    while(1){
        printf("\n1.PUSH 2.POP 3.DISPLAY 4.EXIT \n");
        scanf("%d",&choice);
        switch(choice){
            case 1:{printf("Enter a value:");
                    scanf("%d",&value);
                    push(value);
                    break;}
            case 2:pop();
```

```
break;
case 3:display();
break;
case 4:exit(0);
break;
default:printf("Invalid input");
}
}
return 0;
}
```

Output:

```
Enter the expression : a+b*c-d+h*i
a b c * + d - h i * +
```


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

Code:

```
#include <stdio.h>

#include <stdlib.h>

#define SIZE 5

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

void insert(int value){
    if(front== -1 && rear == -1){
        front=0;
        rear=0;
        q[rear]=value;
    }else if(rear==SIZE-1){
        printf("Overflow");}
    else{
        rear++;
        q[rear]=value;
    }
}

void delete() {
    if (front == -1) {
        printf("Underflow");
    } else {
        if (front > rear) {
            front = -1;
            rear = -1;
        } else {
            printf("%d deleted", q[front]);
            front++;
        }
    }
}
```

```

        }
    }
}

void display(){
if(front==-1){
    printf("Underflow");
}else{
for(int i=front;i<=rear;i++){
    printf("%d",q[i]);
}

}}

int main(){
    int choice,value;
    while(1){
        printf("\n1.INSERT 2.DELETE 3.DISPLAY 4.EXIT:\n");
        scanf("%d",&choice);
        switch(choice){
            case 1:{printf("Enter a value:");
                scanf("%d",&value);
                insert(value);
                break;}
            case 2:delete();
                break;
            case 3:display();
                break;
            case 4:exit(0);
                break;

```

```

        default:printf("Invalid input");

    }

}

return 0;

}

```

Output:

```

1.INSERT 2.DELETE 3.DISPLAY 4.EXIT:
2
Underflow
1.INSERT 2.DELETE 3.DISPLAY 4.EXIT:
1
Enter a value:1

1.INSERT 2.DELETE 3.DISPLAY 4.EXIT:
1
Enter a value:2

1.INSERT 2.DELETE 3.DISPLAY 4.EXIT:
1
Enter a value:3

1.INSERT 2.DELETE 3.DISPLAY 4.EXIT:
1
Enter a value:4

1.INSERT 2.DELETE 3.DISPLAY 4.EXIT:
1
Enter a value:5

1.INSERT 2.DELETE 3.DISPLAY 4.EXIT:
1
Enter a value:6
Overflow
1.INSERT 2.DELETE 3.DISPLAY 4.EXIT:
3
12345
1.INSERT 2.DELETE 3.DISPLAY 4.EXIT:
2
1 deleted

```

Lab Program 3b: WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & Display The program should print appropriate messages for queue empty and queue overflow conditions.

Code:

```
#include <stdio.h>

#include <stdlib.h>

#define SIZE 5

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

void insert(int value){
    if(front== -1 && rear == -1){
        front=0;
        rear=0;
        q[rear]=value;
    }else if((rear+1)%SIZE==front){
        printf("Queue Overflow");}
    else{
        rear=(rear+1)%SIZE;
        q[rear]=value;
    }
}

void delete(){
    if(front== -1){
        printf("Queue Underflow");

    } else{
        if(front==rear){
            front=-1;
            rear=-1;
        }else{
            printf("%d deleted",q[front]);
```

```

        front=(front+1)%SIZE;}

    }

}

void display(){
if(front==-1){
    printf("Queue Underflow");
}else{
    int i=front;
while((i != (rear + 1))){
    printf("%d \t",q[i]);
    i=(i+1)%SIZE;
}

}}

int main(){
    int choice,value;
    while(1){
        printf("\n1.insert 2.delete 3.DISPLAY 4.EXIT:\n");
        scanf("%d",&choice);
        switch(choice){
            case 1:{printf("Enter a value:");
                scanf("%d",&value);
                insert(value);
                break;}
            case 2:delete();
                break;
            case 3:display();
                break;

```

```
        case 4:exit(0);

        break;

        default:printf("Invalid Input");

    }

}

return 0;

}
```

Output:

```
Enter your choice:
1.Insert
2.Delete
3.Display
4.Exit
1
Enter value: 6
Enter your choice:
1.Insert
2.Delete
3.Display
4.Exit
1
Enter value: 5
Enter your choice:
1.Insert
2.Delete
3.Display
4.Exit
2
Value removed is 3
Enter your choice:
1.Insert
2.Delete
3.Display
4.Exit
3
Queue is: 6      5
Enter your choice:
1.Insert
2.Delete
3.Display
4.Exit
4
```

Lab Program 4:

Write a program to implement Singly Linked List with the following operations

- a. Create a linked list.
- b. Insertion of a node at the first position, at any position, and at the end of the list.
- c. deletion of the first element, specified element, and last element in the list.
- d. Display the contents of the linked list.

Code:

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

struct node {
    int data;
    struct node *next;
}*head=NULL;

void insert_first(struct node *p,int x){
    struct node *t;
    t=(struct node *)malloc(sizeof(struct node *));

    t->next=head;
    t->data=x;
    head=t;
}

void insert_last(struct node *p,int x){
    struct node *t;
    t=(struct node *)malloc(sizeof(struct node *));
    t->data=x;
    while(p->next!=NULL){
        p=p->next;
    }
    p->next=t;
    t->next=NULL;
}

void insert_pos(struct node *p,int x,int pos){
    struct node *t,*q;
    t=(struct node *)malloc(sizeof(struct node *));
    t->data=x;
    for (int i = 1; i < pos; i++)
    {
        q=p;
        p=p->next;
    }
    t->next=p;
    q->next=t;
}

void delete_first(struct node *p){
    if(head->next==NULL){
        free(head);
```

```

        return;
    }
    head=head->next;
    free(p);
}

void delete_last(struct node *p){

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

void delete_pos(struct node *p,int pos){

    struct node * q;
    for (int i = 1; i < pos; i++)
    {
        q=p;
        p=p->next;
    }

    q->next=p->next;
    free(p);
}

void display(struct node *p){
    while(p){
        int x= p->data;
        int y=p->next;
        printf("%d(%d)->",x,y);
        p=p->next;
    }
    printf("NULL \n");
}

int main(){

    insert_first(head,5);
    insert_first(head,7);
    insert_last(head,9);
    insert_pos(head,2,2);
    display(head);
    delete_first(head);
    display(head);
    delete_last(head);
    display(head);
    insert_last(head,9);

```



```
display(head);
delete_pos(head,2);
display(head);

return 0;
}
```

Output:

```
      ^
7(11802528)->2(11802792)->5(11802824)->9(0)->NULL
2(11802792)->5(11802824)->9(0)->NULL
2(11802792)->5(0)->NULL
2(11802792)->5(11802808)->9(0)->NULL
2(11802808)->9(0)->NULL
```

Lab Program 5a: Write a program to implement a Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists.

Code:

```
#include<stdio.h>
#include<stdlib.h>
struct node {
    int data;
    struct node *next;
};
void display(struct node *head)
{
    struct node *ptr = head;
    while (ptr != NULL)
    {
        printf("%d\t", ptr->data);
        ptr = ptr->next;
    }
    printf("\n");
}
void sort(struct node **head)
{
    if (*head == NULL)
        return;
    struct node *current, *next;
    int temp;
    current = *head;
    while (current->next != NULL)
    {
        next = current->next;
        while (next != NULL)
        {
            if (current->data > next->data)
            {
                temp = current->data;
                current->data = next->data;
                next->data = temp;
            }
            next = next->next;
        }
        current = current->next;
    }
}
void reverse(struct node **head)
{
    struct node *cur=*head, *prev=NULL, *next=NULL;
    while(cur!=NULL)
    {
        next=cur->next;
```

```

        cur->next=prev;
        prev=cur;
        cur=next;
    }
    *head=prev;
}
struct node *concatenate(struct node **head1, struct node **head2)
{
    if (*head1 == NULL)
    {
        *head1 = *head2;
        return *head1;
    }
    if (*head2 == NULL)
        return *head1;
    struct node *temp = *head1;
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = *head2;
    return *head1;
}
void PUSH(struct node **head)
{
    struct node *node = (struct node*)malloc(sizeof(struct node));
    if (node == NULL)
    {
        printf("Overflow\n");
        exit(1);
    }
    int n;
    printf("Enter value: ");
    scanf("%d", &n);
    node->data = n;
    node->next = *head;
    *head = node;
}
int main()
{
    struct node *head1 = NULL, *head2 = NULL;
    int ch;
    printf("Creating list 1\nEnter no. of elements: ");
    int n, i;
    scanf("%d", &n);
    for (i = 0; i < n; i++)
        PUSH(&head1);
    printf("List 1: ");
    display(head1);
    sort(&head1);
    printf("Sorted list: ");
    display(head1);
}

```

```

reverse(&head1);
printf("Reversed list: ");
display(head1);
printf("Creating list 2\nEnter no. of elements: ");
int n1, i1;
scanf("%d", &n1);
for (i1 = 0; i1 < n1; i1++)
    PUSH(&head2);
printf("List 2: ");
display(head2);
sort(&head2);
printf("Sorted list: ");
display(head2);
reverse(&head2);
printf("Reversed list: ");
display(head2);
printf("Concatenating the 2 lists \n");
struct node *h = concatenate(&head1, &head2);
display(h);
return 0;
}

```

Output:

```

Creating list 1
Enter no. of elements: 5
Enter value: 23
Enter value: 15
Enter value: 28
Enter value: 36
Enter value: 27
List 1: 27    36    28    15    23
Sorted list: 15 23    27    28    36
Reversed list: 36    28    27    23    15
Creating list 2
Enter no. of elements: 5
Enter value: 17
Enter value: 23
Enter value: 34
Enter value: 28
Enter value: 37
List 2: 37    28    34    23    17
Sorted list: 17 23    28    34    37
Reversed list: 37    34    28    23    17
Concatenating the 2 lists
36    28    27    23    15    37    34    28    23    17

```

Program 5b: Write a program to implement a Single Link List to simulate Stack & Queue Operations.

Code:

Stack implementation:

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

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

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

struct Node *push(struct Node *top, int data) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Stack Overflow\n");
        return top;
    }

    newNode->data = data;
    newNode->next = top;
    top = newNode;

    return top;
}

struct Node *pop(struct Node *top, int *poppedData) {
    if (top == NULL) {
        printf("Stack Underflow\n");
        *poppedData = -1; // You can choose another value to indicate underflow
        return NULL;
    }

    struct Node *temp = top;
    *poppedData = temp->data;
    top = top->next;
    free(temp);

    return top;
}
```

```

}

int main() {
    int op, n, poppedElement;
    struct Node *top = NULL;
    printf("Enter 1. Push\n2. Pop\n3. -1 to stop\n");
    while (1) {
        printf("Enter operation:\n");
        scanf("%d", &op);

        if (op == -1) {
            printf("Execution stopped\n");
            break;
        }

        switch (op) {
            case 1:
                printf("Enter the element to push\n");
                scanf("%d", &n);
                top = push(top, n);
                break;
            case 2:
                top = pop(top, &poppedElement);
                if (poppedElement != -1) {
                    printf("Popped Element: %d\n", poppedElement);
                }
            }
            display(top);
        }

        return 0;
    }
}

```

Queue implementation:

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

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

```

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

```

```
void display(struct Node* front) {
```

```

if (front == NULL) {
    printf("Queue is empty\n");
    return;
}
struct Node* temp = front;
printf("Queue elements are:\t");
while (temp != NULL) {
    printf("%d\t", temp->data);
    temp = temp->next;
}
printf("\n");
}

void enqueue(struct Node** front, struct Node** rear, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Queue Overflow\n");
        return;
    }
    newNode->data = data;
    newNode->next = NULL;

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

```

```

int dequeue(struct Node** front, struct Node** rear) {
    if (*front == NULL) {
        printf("Queue Underflow\n");
        return -1; // You can choose another value to indicate underflow
    }
    struct Node* temp = *front;
    int dequeuedData = temp->data;
    *front = (*front)->next;
    if (*front == NULL) {
        *rear = NULL; // If the last element is dequeued, update rear
    }
    free(temp);
    return dequeuedData;
}

```

```

int main() {
    int op, n, dequeuedElement;
    struct Node* front = NULL;
    struct Node* rear = NULL;
    printf("Enter 1. Enqueue\n2. Dequeue\n3. -1 to stop\n");
    while (1) {
        printf("Enter operation\n");
        scanf("%d", &op);
        if (op == -1) {
            printf("Execution stopped\n");
            break;
        }
        switch (op) {
            case 1:

```



```

printf("Enter the element to enqueue\n");

scanf("%d", &n);

enqueue(&front, &rear, n);

break;

case 2:

dequeuedElement = dequeue(&front, &rear);

if (dequeuedElement != -1) {

    printf("Dequeued Element: %d\n", dequeuedElement);

}

break;

}

display(front);

}

return 0;

}

```

Output:

| | |
|---------------------------|------------------------------|
| Enter 1. Push | Enter 1. Enqueue |
| 2. Pop | 2. Dequeue |
| 3. -1 to stop | 3. -1 to stop |
| Enter operation: | Enter operation |
| 1 | 1 |
| Enter the element to push | Enter the element to enqueue |
| 12 | 25 |
| Stack elements are: 12 | Queue elements are: 25 |
| Enter operation: | Enter operation |
| 1 | 1 |
| Enter the element to push | Enter the element to enqueue |
| 15 | 67 |
| Stack elements are: 15 12 | Queue elements are: 25 67 |
| Enter operation: | Enter operation |
| 2 | 2 |
| Popped Element: 15 | Dequeued Element: 25 |
| Stack elements are: 12 | Queue elements are: 67 |
| Enter operation: | Enter operation |
| 2 | 2 |
| Popped Element: 12 | Dequeued Element: 67 |
| Stack is empty | Queue is empty |
| Enter operation: | Enter operation |
| 2 | 2 |
| Stack Underflow | Queue Underflow |
| Stack is empty | Queue is empty |
| Enter operation: | Enter operation |
| -1 | -1 |
| Execution stopped | Execution stopped |

Lab Program 6: Write a program to Implement doubly link list with primitive operations

- e. Create a doubly linked list.**
- f. Insert a new node to the left of the node.**
- g. Delete the node based on a specific value**
- h. Display the contents of the list**

Code:

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

struct node
{
    struct node *next, *prev;
    int data;
};
struct node *head;
void display()
{
    struct node *temp = head;
    if (temp == NULL)
    {
        printf("List is empty.\n");
        return;
    }
    while (temp != NULL)
    {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
void push()
{
    struct node *new_node = (struct node *)malloc(sizeof(struct node));
    int data;
    if(new_node==NULL)
        printf("Overflow\n");
    else
    {
        printf("Enter the data: ");
        scanf("%d", &data);
        new_node->data = data;
        if (head == NULL)
        {
            new_node->next = NULL;
            new_node->prev = NULL;
            head = new_node;
        }
    }
}
```

```

    }
    else
    {
        head->prev = new_node;
        new_node->next = head;
        new_node->prev = NULL;
        head = new_node;
    }
}

void delete_specified()
{
    int loc=1, val;
    printf("Enter the value to delete: ");
    scanf("%d", &val);
    struct node *temp = head;
    if (temp == NULL)
    {
        printf("List is empty. Nothing to delete.\n");
        return;
    }
    while(temp->data!=val)
    {
        loc++;
        temp=temp->next;
    }
    temp=head;
    if (loc == 1)
    {
        head = temp->next;
        if (head != NULL)
            head->prev = NULL;
        free(temp);
        printf("Node deleted from the beginning.\n");
        return;
    }
    for (int i = 1; i < loc; i++)
    {
        temp = temp->next;
        if (temp == NULL)
        {
            printf("Specified position does not exist.\n");
            return;
        }
    }
    if (temp->next == NULL)
    {
        temp->prev->next = NULL;
        free(temp);
    }
}

```

```

        printf("Node deleted from the end.\n");
        return;
    }
    temp->prev->next = temp->next;
    temp->next->prev = temp->prev;
    free(temp);
    printf("Node deleted from location %d.\n", loc);
    printf("After Deletion: ");
    display();
}
int main()
{
    int ch;
    while (1)
    {
        printf("Enter your choice:\n1.Insert a new node\n2.Delete a node\n3.Display the
list\n4.Exit\n");
        scanf("%d", &ch);
        switch (ch)
        {
            case 1:{
                printf("Enter no. of elements: ");
                int n, i;
                scanf("%d",&n);
                for(i=0; i<n; i++)
                    push();
                printf("After insertion: ");
                display();
            }break;
            case 2:{
                printf("Enter no. of elements to delete: ");
                int n,i;
                scanf("%d",&n);
                for(i=0; i<n; i++)
                    delete_specified();
            }break;
            case 3:display(); break;
            case 4:exit(0);
        }
    }
    return 0;
}

```

Output:

```
Enter your choice:
1.Insert a new node
2.Delete a node
3.Display the list
4.Exit
1
Enter no. of elements: 4
Enter the data: 10
Enter the data: 20
Enter the data: 30
Enter the data: 40
After insertion: 40 30 20 10
Enter your choice:
1.Insert a new node
2.Delete a node
3.Display the list
4.Exit
3
40 30 20 10
Enter your choice:
1.Insert a new node
2.Delete a node
3.Display the list
4.Exit
2
Enter no. of elements to delete: 2
Enter the value to delete: 30
Node deleted from location 2.
After Deletion: 40 20 10
Enter the value to delete: 10
Node deleted from the end.
Enter your choice:
1.Insert a new node
2.Delete a node
3.Display the list
4.Exit
3
40 20
Enter your choice:
1.Insert a new node
2.Delete a node
3.Display the list
4.Exit
4
```

Leetcode-856

Code:

```
int scoreOfParentheses(char* s) {
    int n=strlen(s),ans=0;
    int d=0,i=0;
    while(i<n) {
        if(s[i]=='(') d++;
        else {
```

```

        d--;

        if(i>0 && s[i-1]=='(') ans+=1<<d;
    }

    i++;
}

return ans;
}

```

Output:

Accepted

Varnith31 submitted at Mar 03, 2024 17:05

Editorial

Solution

Runtime

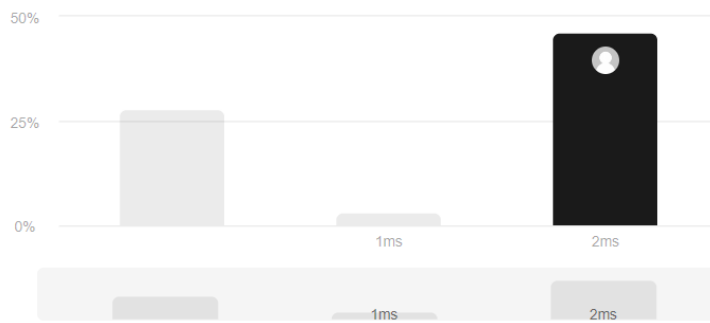
2 ms

Beats 69.23% of users with C

Memory

5.72 MB

Beats 20.00% of users with C



Code | C

Lab Program 7: 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
To display the elements in the tree.

Code:

```
#include<stdio.h>
#include<stdlib.h>
typedef struct BST
{
    int data;
    struct BST *left;
    struct BST *right;
}node;
node *create()
{
    node *t;
    printf("Enter data: ");
    t=(node*)malloc(sizeof(node));
    scanf("%d",&t->data);
    t->left=t->right=NULL;
    return t;
}
void insert(node *root,node*t)
{
    if(t->data<root->data)
    {
        if(root->left!=NULL)
            insert(root->left,t);
        else
            root->left=t;
    }
    if(t->data>root->data)
    {
        if(root->right!=NULL)
            insert(root->right,t);
        else
            root->right=t;
    }
}
void preorder(node *root)
{
    if(root!=NULL)
    {
        printf("%d ",root->data);
        preorder(root->left);
        preorder(root->right);
    }
}
```

```

void inorder(node *root)
{
    if(root!=NULL)
    {
        inorder(root->left);
        printf("%d ",root->data);
        inorder(root->right);
    }
}
void postorder(node *root)
{
    if(root!=NULL)
    {
        postorder(root->left);
        postorder(root->right);
        printf("%d ",root->data);
    }
}
int main()
{
    char ch;
    node *root=NULL,*t;
    do{
        t=create();
        if(root==NULL)
            root=t;
        else
            insert(root,t);
        printf("Do you want to enter more?(y/n) ");
        getchar();
        scanf("%c",&ch);
    }while(ch=='y'||ch=='Y');
    int c;
    while(1)
    {
        printf("\nEnter your choice:\n1.Preorder\n2.Inorder\n3.Postorder\n4.Exit\n");
        scanf("%d",&c);
        switch(c)
        {
            case 1:{preorder(root);}break;
            case 2:{inorder(root);}break;
            case 3:{postorder(root);}break;
            case 4:exit(0);
        }
    }
    return 0;
}

```


Output:

```
Enter data: 10
Do you want to enter more?(y/n) y
Enter data: 5
Do you want to enter more?(y/n) y
Enter data: 15
Do you want to enter more?(y/n) y
Enter data: 2
Do you want to enter more?(y/n) y
Enter data: 8
Do you want to enter more?(y/n) y
Enter data: 13
Do you want to enter more?(y/n) y
Enter data: 20
Do you want to enter more?(y/n) n

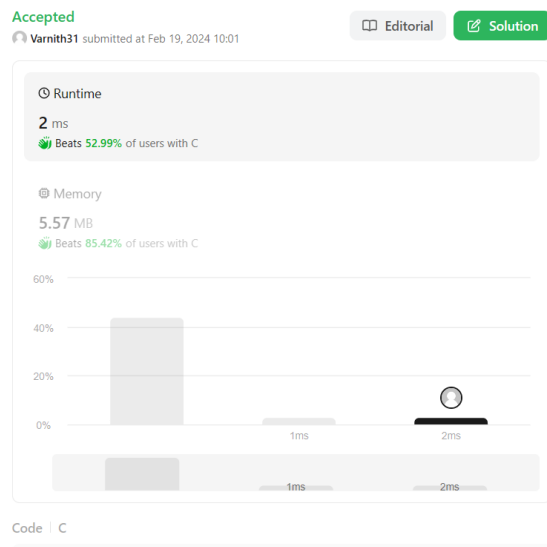
Enter your choice:
1.Preorder
2.Inorder
3.Postorder
4.Exit
1
10 5 2 8 15 13 20
Enter your choice:
1.Preorder
2.Inorder
3.Postorder
4.Exit
2
2 5 8 10 13 15 20
Enter your choice:
1.Preorder
2.Inorder
3.Postorder
4.Exit
3
2 8 5 13 20 15 10
Enter your choice:
1.Preorder
2.Inorder
3.Postorder
4.Exit
4
```

Leetcode-876

Code:

```
struct ListNode* deleteMiddle(struct ListNode* head) {  
    if (head == NULL) return NULL;  
    struct ListNode* prev = (struct ListNode*)malloc(sizeof(struct ListNode));  
    prev->val = 0;  
    prev->next = head;  
    struct ListNode* slow = prev;  
    struct ListNode* fast = head;  
    while (fast != NULL && fast->next != NULL) {  
        slow = slow->next;  
        fast = fast->next->next;  
    }  
    struct ListNode* temp = slow->next;  
    slow->next = slow->next->next;  
    free(temp);  
    struct ListNode* newHead = prev->next;  
    free(prev);  
    return newHead;  
}
```

Output:



Leetcode-328:

Code:

```
struct ListNode* oddEvenList(struct ListNode* head) {  
    if(head==NULL || head->next==NULL)  
        return head;  
  
    struct ListNode* oddH = NULL, *oddT = NULL, *evenH = NULL, *evenT = NULL;  
  
    struct ListNode* curr = head;  
  
    int i = 1;  
    while(curr != NULL){  
        if(i%2 != 0){  
            if(oddH == NULL){  
                oddH = curr;  
                oddT = curr;  
            }  
            else{  
                oddT -> next = curr;  
                oddT = curr;  
            }  
        }  
        curr = curr->next;  
        i++;  
    }  
    oddT->next = NULL;  
    return oddH;  
}
```


```
}  
else{  
    if(evenH == NULL){  
        evenH = curr;  
        evenT = curr;  
    }  
    else{  
        evenT -> next = curr;  
        evenT = curr;  
    }  
}  
i++;  
curr = curr -> next;  
}  
evenT -> next = NULL;  
oddT -> next = NULL;  
oddT->next = evenH;  
return oddH;  
}
```

Output:

Accepted


 Varnith31 submitted at Feb 19, 2024 10:38

 Editorial

 Solution

Runtime

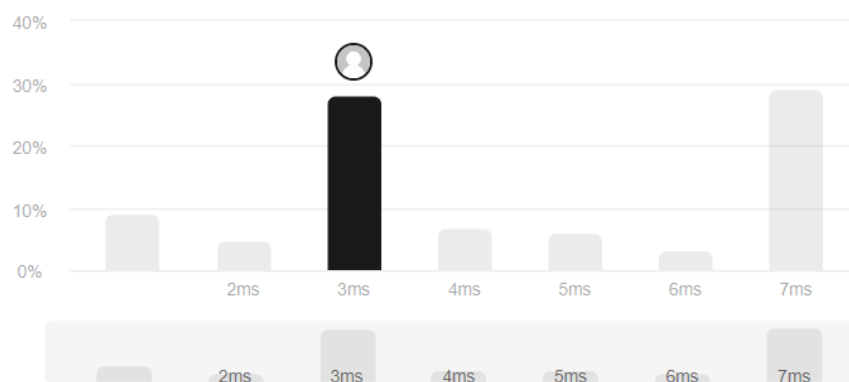
3 ms

 Beats 86.05% of users with C

Memory

6.54 MB

 Beats 94.45% of users with C



Code | C

1..

Lab Program 8a: Write a program to traverse a graph using the BFS method.

Code:

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

#define MAX_VERTICES 50

typedef struct Graph_t {

    int V;
    bool adj[MAX_VERTICES][MAX_VERTICES];
} Graph;

Graph* Graph_create(int V)
{
    Graph* g = malloc(sizeof(Graph));
    g->V = V;

    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            g->adj[i][j] = false;
        }
    }

    return g;
}

void Graph_addEdge(Graph* g, int v, int w)
{
    // Add w to v's list.
    g->adj[v][w] = true;
}

void Graph_BFS(Graph* g, int s)
{
    bool visited[MAX_VERTICES];
    for (int i = 0; i < g->V; i++) {
        visited[i] = false;
    }

    int queue[MAX_VERTICES];
    int front = 0, rear = 0;

    visited[s] = true;
    queue[rear++] = s;

    while (front != rear) {
```

```

s = queue[front++];
printf("%d ", s);

for (int adjacent = 0; adjacent < g->V;
adjacent++) {
if (g->adj[s][adjacent] && !visited[adjacent]) {
visited[adjacent] = true;
queue[rear++] = adjacent;
}
}
}
}
}

```

```

int main()
{

Graph* g = Graph_create(4);
Graph_addEdge(g, 0, 1);
Graph_addEdge(g, 0, 2);
Graph_addEdge(g, 1, 2);
Graph_addEdge(g, 2, 0);
Graph_addEdge(g, 2, 3);
Graph_addEdge(g, 3, 3);

printf("Following is Breadth First Traversal "
"(starting from vertex 2) \n");
Graph_BFS(g, 2);

return 0;
}

```

Output:

```

Following is Breadth First Traversal (starting from vertex 2)
2 0 3 1

```

Lab Program 8b: Write a program to check whether a graph is connected or not using the DFS method.

Code:

```
#include <stdio.h>

#define MAX_NODES 100

int visited[MAX_NODES];
int adj_matrix[MAX_NODES][MAX_NODES];
int num_nodes;

void dfs(int vertex) {
    visited[vertex] = 1;
    printf("%d ", vertex);

    for (int i = 0; i < num_nodes; i++) {
        if (adj_matrix[vertex][i] && !visited[i]) {
            dfs(i);
        }
    }
}

int main() {
    printf("Enter the number of nodes: ");
    scanf("%d", &num_nodes);

    printf("Enter the adjacency matrix:\n");
    for (int i = 0; i < num_nodes; i++) {
        for (int j = 0; j < num_nodes; j++) {
            scanf("%d", &adj_matrix[i][j]);
        }
    }
    for (int i = 0; i < num_nodes; i++) {
        visited[i] = 0;
    }

    printf("DFS Traversal: ");
    for (int i = 0; i < num_nodes; i++) {
        if (!visited[i]) {
            dfs(i);
        }
    }
    printf("\n");

    return 0;
}
```


Output:

```
Enter the number of nodes: 4
Enter the adjacency matrix:
0 1 0 0
0 0 1 1
0 1 0 1
0 1 1 0
DFS Traversal: 0 1 2 3
```

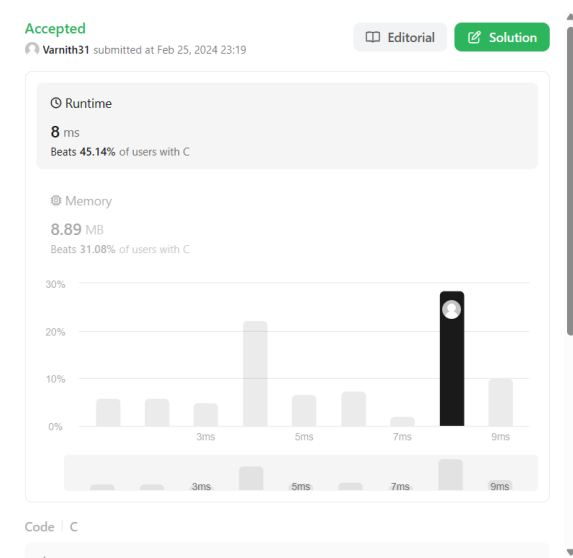
Leetcode-513

Code:

```
int findBottomLeftValue(struct TreeNode* root) {
    int value=root->val;
    int mdepth=0;

    void transverse(struct TreeNode* p,int depth){
        if(!p)
            return;
        if(depth>mdepth){
            mdepth=depth;
            value=p->val;
        }
        transverse(p->left,depth+1);
        transverse(p->right,depth+1);
    }
    transverse(root,0);
    return value;
}
```

Output:



Leetcode-450

Code:

```
struct TreeNode* deleteNode(struct TreeNode* root, int key) {  
    if (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 && !root->right)  
                return NULL;  
            if (!root->left || !root->right)  
                return root->left ? root->left : root->right;  
  
            struct TreeNode* temp = root->left;  
            while (temp->right != NULL)  
                temp = temp->right;  
            root->val = temp->val;  
        }  
    }  
}
```

```

        root->left = deleteNode(root->left, temp->val);

    }

}

return root;

}

```

Output:

