VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

VEMULA DHANUSH REDDY (1BM22CS324)

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 **VEMULA DHANUSH REDDY (1BM22CS324)**, 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.

Surabhi S
Assistant Professor
Department of CSE
BMSCE, Bengaluru

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

Index Sheet

5	**Program 5:** **Write a program 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 **Leetcode Program #856 :** Score of Parenthesis	27-30
6	**Program 6:** Write a program 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 program #2095 :** Delete Middle Node of a LinkedList **Leetcode program #328 :** Odd Even Linked list	31-34
7	**Program 7a:** Write a program to traverse a graph using the BFS method. **Program 7b:** Write a program to check whether a graph is connected or not using the DFS method. **Leetcode Program #513:** Find Bottom Left Tree Value **Leetcode Program #450 :** Delete Node in a BST	35-41

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.

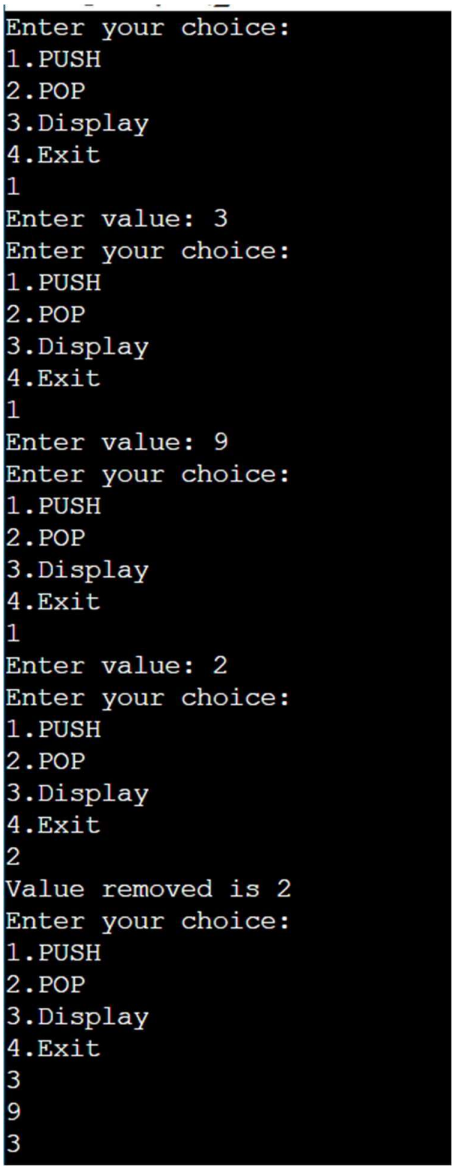
Lab Program 1a: Write a program to implement basic stack operations such as PUSH, POP, and Display.

Code:

```
#include<stdio.h>
#include<ctype.h>
#include <stdlib.h>
#define s 5
int stack[s];
int t=-1;

void PUSH(int a)
{
   if(t==s-1)
      printf("Stack is full");
   else
   {
      t++;
      stack[t]=a;
   }
}
void POP()
{
   if(t==-1)
      printf("Stack is empty");
   else
   {
      printf("Value removed is %d\n",stack[t]);
      t--;
   }
}
void display()
{
   int i;
   for(i=t; i>=0;i--)
      printf("%d\n",stack[i]);
}
int main()
{
   int ch;
   do
   {
      printf("Enter your choice:\n1.PUSH\n2.POP\n3.Display\n4.Exit\n");
      scanf("%d",&ch);
      switch(ch)
      {
         case 1: {
                  printf("Enter value: ");
                  int n;
```

```
                scanf("%d",&n);
                PUSH(n);
            }break;
        case 2: POP(); break;
        case 3: display(); break;
        case 4: exit(0);
      }
   }while(1);
   return 0;
}
```

Output:



Lab Program 1b:

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 <ctype.h>
#define SIZE 50
char stack[SIZE];
int top=-1;

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

char pop()
{
        return(stack [top--]);
}
int pr(char symbol)
{
        if (symbol == '^')
        {
                return(3);
        }
        else if(symbol=='*' || symbol=='/')
        {
                return(2);
        }
        else if(symbol == '+' || symbol == '-')
        {
                return(1);
        }
        else
        {
                return(0);
        }
}
void main()
{
        char infix[50], postfix[50],ch,elem;
        int i=0,k=0;
        printf("Enter Infix Expression : ");
        scanf("%s", infix);
        push('#');
        while ((ch=infix[i++]) != '\0')
        {
```

```
            if( ch == '(')
                    push(ch);
            else if(isalnum(ch))
                    postfix[k++]=ch;
            else
            {
                    if( ch == ')')
                    {
                        while (stack[top] != '(')
                        postfix[k++]=pop();
                        elem=pop();
                    }
                    else
                    {
                       while( pr(stack [top]) >= pr(ch) )
                       postfix [k++]=pop();
                       push(ch);
                    }
            }
        }
        while (stack[top] != '#')
                 postfix[k++]=pop();
        postfix[k]='\0';
        printf("\nPostfix Expression = %s\n", postfix);
}
```

Output:

```
Enter Infix Expression : A+B*C-D+H*I

Postfix Expression = ABC*+D-HI*+
```

Lab Program 2a: 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<ctype.h>
#include <stdlib.h>
#define s 5
int queue[s], f=-1, r=-1;

int isfull()
{
   if(f==(s-1))
      return 1;
   else
      return 0;
}
int isempty()
{
   if(f==-1||r==-1)
      return 1;
   else
      return 0;
}
void insert()
{
   int i;
   printf("Enter value: ");
   scanf("%d",&i);
   if(isfull())
      printf("Queue is full");
   else if(f==-1)
   {
      f=0;
      r=0;
   }
   else
      r=r+1;
   queue[r]=i;
}
void qdelete()
{
   if(isempty())
      printf("Queue is empty");
   else if(f==r)
   {
      f=-1;
      r=-1;
   }
```

```
    else
    {
        printf("Value removed is %d\n",queue[f]);
        f=f+1;
    }
}
void display()
{
    int i;
    if(isempty())
        printf("Queue is empty");
    else
    {
        printf("Queue is: ");
        for(i=f; i<=r;i++)
            printf("%d\t",queue[i]);
        printf("\n");
    }
}
int main()
{
    int ch;
    while(1)
    {
        printf("Enter your choice:\n1.Insert\n2.Delete\n3.Display\n4.Exit\n");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1: insert(); break;
            case 2: qdelete(); break;
            case 3: display(); break;
            case 4: {
                        printf("Name: Amrutha Ravi \tUSN:1BM22CS036");
                        exit(0);
                    }
        }
    }
    return 0;
}
```

Output:

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

Process returned 0 (0x0)   execution time : 11.359 s
Press any key to continue.
```

Lab Program 2b: 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<ctype.h>
#include <stdlib.h>
#define s 5
int queue[s], f=-1, r=-1;

int isfull()
{
   if(f==(r+1)||f==0 && r==(s-1))
      return 1;
   else
      return 0;
}
int isempty()
{
   if(f==-1||f>r)
      return 1;
   else
      return 0;
}
void insert()
{
   int i;
   printf("Enter value: ");
   scanf("%d",&i);
   if(isfull())
      printf("Queue is full");
   else if(isempty())
   {
      f=0;
      r=0;
   }
   else
      r=(r+1)%s;
   queue[r]=i;
}
void qdelete()
{
   if(isempty())
      printf("Queue is empty");
   else if(f==r)
   {
      f=-1;
      r=-1;
```

```
    }
    else
    {
        printf("Value removed is %d\n",queue[f]);
        f=(f+1)%s;
    }
}
void display()
{
    int i;
    if(isempty())
        printf("Queue is empty");
    else
    {
        printf("Queue is: ");
        for(i=f; i!=r;i=(i+1)%s)
            printf("%d\t",queue[i]);
        printf("%d",queue[i]);
        printf("\n");
    }
}
int main()
{
    int ch;
    while(1)
    {
        printf("Enter your choice:\n1.Insert\n2.Delete\n3.Display\n4.Exit\n");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1: insert(); break;
            case 2: qdelete(); break;
            case 3: display(); break;
            case 4: {
                        exit(0);
                    }
        }
    }
    return 0;
}
```

Output:

```
Enter your choice:
1.Insert
2.Delete
3.Display
4.Exit
1
Enter value: 3
Enter your choice:
1.Insert
2.Delete
3.Display
4.Exit
3
Queue is: 5     7       3
Enter your choice:
1.Insert
2.Delete
3.Display
4.Exit
2
Value removed is 5
Enter your choice:
1.Insert
2.Delete
3.Display
4.Exit
1
Enter value: 7
Enter your choice:
1.Insert
2.Delete
3.Display
4.Exit
3
Queue is: 7     3       7
Enter your choice:
1.Insert
2.Delete
3.Display
4.Exit
4
```

Lab Program 3a: Write a program to implement Singly Linked List with the following operations

d. **Create a linked list.**
e. **Insertion of a node at the first position, at any position, and at the end of the list.**
f. **Display the contents of the linked list.**

Code:

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

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

struct node *createnode(int val) {
   struct node *newnode = (struct node*)malloc(sizeof(struct node));
   newnode->data = val;
   newnode->next = NULL;
   return newnode;
}

void insert_beg(struct node **head, int val) {
   struct node *newnode = createnode(val);
   newnode->next = *head;
   *head = newnode;
}

void insert_end(struct node *head, int val) {
   struct node *newnode = createnode(val);
   struct node *temp = head;
   while (temp->next != NULL) {
      temp = temp->next;
   }
   temp->next = newnode;
}

void insert_at_pos(struct node *head, int val, int pos) {
   struct node *newnode = createnode(val);
   struct node *temp = head;
   for (int i = 1; i < pos - 1 && temp != NULL; i++) {
      temp = temp->next;
   }
   if (temp != NULL) {
      newnode->next = temp->next;
      temp->next = newnode;
   } else {
      printf("Invalid Position\n");
   }
```

```
}

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

int main() {
   struct node *head = NULL;
   insert_beg(&head, 1);
   display(head);
   insert_end(head, 3);
   display(head);
   insert_at_pos(head, 2, 2);
   display(head);
   return 0;
}
```

Output:

```
1 NULL
1 3 NULL
1 2 3 NULL

Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

Lab Program 3b: Write a program to implement a Singly Linked List with the following operations: deletion of the first element, specified element, and last element in the list.

Code:

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

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

struct node *createnode(int val) {
   struct node *newnode = (struct node*)malloc(sizeof(struct node));
   newnode->data = val;
   newnode->next = NULL;
   return newnode;
}

void insert_beg(struct node **head, int val) {
   struct node *newnode = createnode(val);
   newnode->next = *head;
   *head = newnode;
}

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

void delete_beg(struct node **head)
{
   if(*head==NULL)
   {
      printf("Empty list");
   }
   else
   {
      struct node *temp=*head;
      *head=(*head)->next;
      free(temp);
   }
}
```

```
void delete_end(struct node **head)
{
   if(*head==NULL)
   {
      printf("Empty list");
   }
   else if ((*head)->next==NULL)
   {
      free(*head);
      return NULL;
   }
   else
   {
      struct node*temp=*head;
      struct node*p=NULL;
      while(temp->next!=NULL)
      {
         p=temp;
         temp=temp->next;
      }
      free(temp);
      p->next=NULL;
   }
}

void delete_val(struct node **head,int val)
{
   struct node *temp=*head;
   struct node* p= NULL;
   while(temp!=NULL&&temp->data!=val)
   {
      p=temp;
      temp=temp->next;
   }
   if(temp!=NULL)
   {
      if(p==NULL)
      {
         *head=(*head)->next;
      }
      else
      {
         p->next=temp->next;
      }
      free(temp);
   }
   else{
      printf("Element %d not found",val);
   }
}
```

```
int main() {
   struct node *head = NULL;
   insert_beg(&head, 5);
   insert_beg(&head,  4);
   insert_beg(&head,  3);
   insert_beg(&head,  2);
   insert_beg(&head,  1);

   display(head);

   delete_beg(&head);
   delete_end(&head);
   delete_val(&head,3);
   display(head);
   return 0;
```

Output:

```
1 2 3 4 5 NULL
2 4 NULL

Process returned 0 (0x0)   execution time : 0.030 s
Press any key to continue.
```

Lab Program 4a: 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:

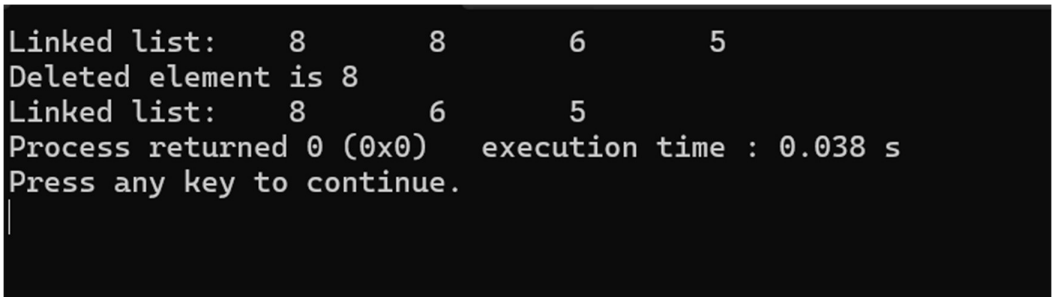
```
Creating list 1
Enter no. of elements: 3
Enter value: 2
Enter value: 4
Enter value: 6
List 1: 6       4       2
Sorted list: 2  4       6
Reversed list: 6        4       2
Creating list 2
Enter no. of elements: 3
Enter value: 3
Enter value: 5
Enter value: 7
List 2: 7       5       3
Sorted list: 3  5       7
Reversed list: 7        5       3
Concatenating the 2 lists
6       4       2       7       5       3
```

Program 4b: Write a program to implement a Single Link List to simulate Stack Operations.

Code:

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

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

void push(int x)
{
   struct node * newnode=(struct node *)malloc(sizeof(struct node));
   newnode->data=x;
   newnode->next=top;
   top=newnode;
}

void display()
{
   struct node *temp=top;
   printf("Linked list: ");
   while(temp!=NULL)
   {
      printf("\t%d",temp->data);
      temp=temp->next;
   }
}

void pop()
{
   struct node *temp=top;
   printf("\nDeleted element is %d\n",temp->data);
   top=top->next;
   free(temp);
}
int main()
{
   push(5);
   push(6);
   push(8);
   push(8);
   display();
   pop();
   display();
}
```

Output:


Linked list: 8 8 6 5
Deleted element is 8
Linked list: 8 6 5
Process returned 0 (0x0) execution time : 0.038 s
Press any key to continue.

Program 4c: Write a program to implement a Single Link List to simulate Queue Operations.

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

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

struct node * front=0;
struct node * rear=0;

void enqueue(int x)
{
   struct node * newnode = (struct node*)malloc(sizeof(struct node));
   newnode->data=x;
   newnode->next=NULL;
   if(front==0&&rear==0)
   {
      front=rear=newnode;
   }
   else
   {
      rear->next=newnode;
      rear=newnode;
   }
}

void displayq()
{
   struct node * temp=front;
   while(temp!=0)
   {
      printf("\t%d",temp->data);
      temp=temp->next;
   }
}
void dequeue()
{
   struct node * temp;
   if(front==0&&rear==0)
   {
      printf("Empty");
   }
   else
   {
      temp=front;
```

```
        front=front->next;
        printf("\nDeleted element is %d\n",temp->data);
        free(temp);
    }
}
int main()
{
    enqueue(5);
    enqueue(6);
    enqueue(7);
    displayq();
    dequeue();
    displayq();
}
```

Output:

```
Queue :         5       6       7
Deleted element is : 5
Queue :         6       7
Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

Lab Program 5: 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 {
   int data;
   struct Node* prev;
   struct Node* next;
};

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

void insertLeft(struct Node** head, int value, int targetValue) {
   struct Node* newNode = createNode(value);

   if (*head == NULL) {
      *head = newNode;
      return;
   }

   struct Node* current = *head;

   while (current != NULL && current->data != targetValue)
   {
      current = current->next;
   }

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

```
        newNode->next = current;
        current->prev = newNode;
    } else {
        printf("Node with value %d not found.\n", targetValue);
    }
}

void deleteNode(struct Node** head, int value) {
    if (*head == NULL) {
        printf("List is empty.\n");
        return;
    }

    struct Node* current = *head;


    while (current != NULL && current->data != value) {
        current = current->next;
    }

    if (current != NULL) {
        if (current->prev != NULL) {
            current->prev->next = current->next;
        } else {
            *head = current->next;
        }

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

        free(current);
        printf("Node with value %d deleted.\n", value);
    } else {
        printf("Node with value %d not found.\n", value);
    }
}

void display(struct Node* head) {
    struct Node* current = head;

    while (current != NULL) {
        printf("%d <-> ", current->data);
        current = current->next;
    }

    printf("NULL\n");
}
```

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


    insertLeft(&head, 3, 0);
    insertLeft(&head, 2, 3);
    insertLeft(&head, 1, 2);

    printf("Initial list: ");
    display(head);

    insertLeft(&head, 4, 3);
    printf("List after insertion: ");
    display(head);


    deleteNode(&head, 2);
    printf("List after deletion: ");
    display(head);

    return 0;
}
```

Output:

```
Initial list: 1 <-> 2 <-> 3 <-> NULL
List after insertion: 1 <-> 2 <-> 4 <-> 3 <-> NULL
Node with value 2 deleted.
List after deletion: 1 <-> 4 <-> 3 <-> NULL

Process returned 0 (0x0)   execution time : 0.013 s
Press any key to continue.
```

Leetcode problem #856: Score Of Parenthesis

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:

Lab Program 6: Write a program

a. **To construct a binary Search tree.**
b. **To traverse the tree using all the methods i.e., in-order, preorder and post order.**
c. **To display the elements in the tree.**

Code:

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

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


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


struct Node *insert(struct Node *root, int value) {
   if (root == NULL) {
      return createNode(value);
   }

   if (value < root->data) {
      root->left = insert(root->left, value);
   } else if (value > root->data) {
      root->right = insert(root->right, value);
   }

   return root;
}


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


void preorderTraversal(struct Node *root) {
```

```
    if (root != NULL) {
        printf("%d ", root->data);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
    }
}


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


void display(struct Node *root) {
    printf("In-order Traversal: ");
    inorderTraversal(root);
    printf("\n");

    printf("Pre-order Traversal: ");
    preorderTraversal(root);
    printf("\n");

    printf("Post-order Traversal: ");
    postorderTraversal(root);
    printf("\n");
}

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


    root = insert(root, 50);
    insert(root, 30);
    insert(root, 20);
    insert(root, 40);
    insert(root, 70);
    insert(root, 60);
    insert(root, 80);

    display(root);

    return 0;
}
```

Output:

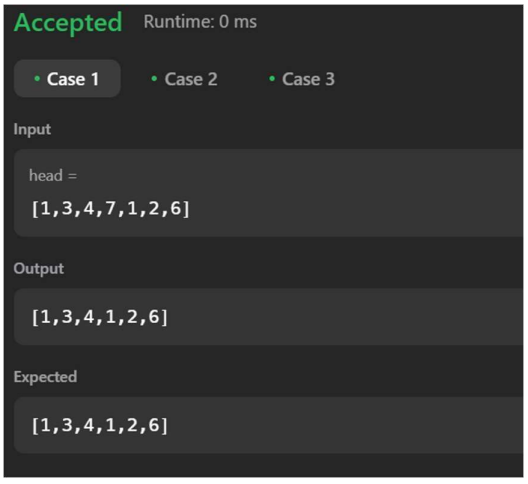
```
In-order Traversal: 20 30 40 50 60 70 80
Pre-order Traversal: 50 30 20 40 70 60 80
Post-order Traversal: 20 40 30 60 80 70 50

Process returned 0 (0x0)   execution time : 0.032 s
Press any key to continue.
```

Leetcode Program 2095 : Delete the middle node of a linked list

Code:

```
 Definition for singly-linked list.
 struct ListNode {
   int val;
   struct ListNode *next;
 };
 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:

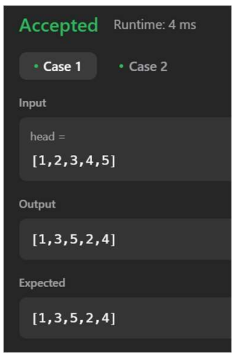


Leetcode Program 328: Odd Even Linked List

Code:

```
 Definition for singly-linked list.
 struct ListNode {
     int val;
     struct ListNode *next;
 };
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;
            }
         }
         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:



Lab Program 7a: Write a program to traverse a graph using the BFS method.

Code:

```
#include<stdio.h>
#include<conio.h>
void bfs(int a[20][20], int n, int src, int t[20][2], int s[])
{
	int f,r,q[20],u,v,k=0,i;
	for(i=1;i<=n;i++)
		s[i]=0;
	f=r=k=0;
	q[r]=src;
	s[src]=1;
	while(f<=r)
	{
		u=q[f++];
		for(v=1;v<=n;v++)
		{
			if(a[u][v]==1 && s[v]==0)
			{
				s[v]=1;
				q[++r]=v;
				t[k][0]=u;
				t[k][1]=v;
				k++;
			}
		}
	}
}
void main()
{
	int n,a[20][20],src,t[20][2],flag,s[20],i,j;
	printf("Enter the number of nodes\n");
	scanf("%d", &n);
	printf("Enter the adjacency matrix\n");
	for(i=0;i<n;i++)
	{
		for(j=0;j<n;j++)
			scanf("%d", &a[i][j]);
	}
	printf("Enter the source\n");
	scanf("%d", &src);
	bfs(a,n,src,t,s);
	flag=0;
	for(i=0;i<n;i++)
	{
		if(s[i]==0)
		{
			printf("Vertex %d is not reachable\n", i);
```

```
                flag=1;
            }
            else
                printf("Vertex %d is reachable\n", i);
        }
        if(flag==1)
            printf("Some nodes are not visited\n");
        else
        {
            printf("The BFS traversal is\n");
            for(i=0;i<n;i++)
                printf("%d%d\n", t[i][0], t[i][1]);
        }
        getch();
}
```

Output:

```
Enter the number of nodes
4
Enter the adjacency matrix
1 0 1 1
1 0 0 1
1 1 0 1
1 1 1 0
Enter the source
0
Vertex 0 is reachable
Vertex 1 is reachable
Vertex 2 is reachable
Vertex 3 is reachable
The BFS traversal is
02
03
21
00
```

Lab Program 7b: Write a program to check whether a graph is connected or not using the DFS method.

Code:

```
#include<stdio.h>
#include<conio.h>
int a[1][10];
void dfs(int n, int cost[10][10], int u, int s[])
{
        int v;
        s[u]=1;
        for(v=0;v<n;v++)
        {
                if((cost[u][v]==1) && (s[v]==0))
                        dfs(n,cost,v,s);
        }
}
void main()
{
        int n,i,j,cost[10][10],s[10],con,flag;
        printf("Enter the number of nodes\n");
        scanf("%d", &n);
        printf("Enter the adjacency matrix\n");
        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                        scanf("%d", &cost[i][j]);
        }
        con=0;
        for(j=0;j<n;j++)
        {
                for(i=0;i<n;i++)
                s[i]=0;
                dfs(n,cost,j,s);
                flag=0;
                for(i=0;i<n;i++)
                {
                        if(s[i]==0)
                                flag=1;
                }
                if(flag==0)
                        con=1;
        }
        if(con==1)
                printf("Graph is connected\n");
        else
                printf("Graph is not connected\n");
        getch();
}
```

Output:

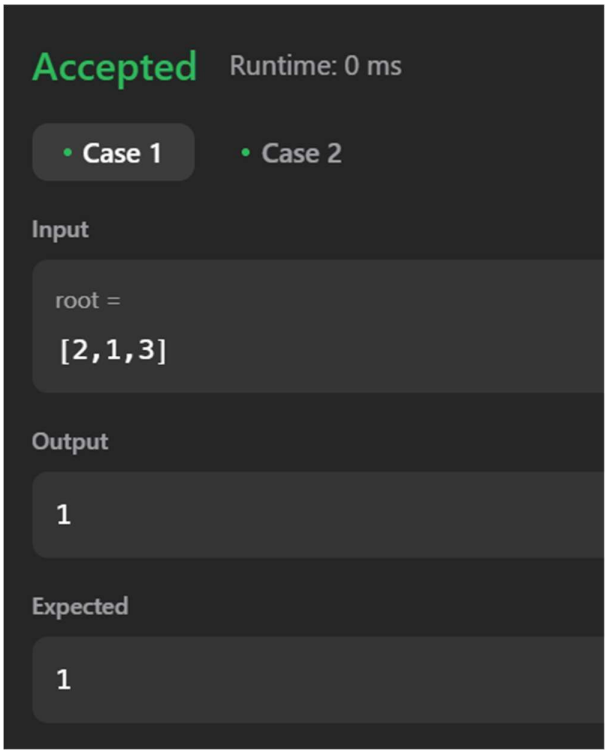
```
Enter the number of nodes
4
Enter the adjacency matrix
1 0 1 1
1 1 0 1
1 1 1 0
1 0 1 1
Graph is connected
```

Leetcode program 513 : Find bottom left Tree value

Code:

```
struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
 };
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:


Accepted
Runtime: 0 ms
• Case 1
• Case 2
Input
root =
[2,1,3]
Output
1
Expected
1

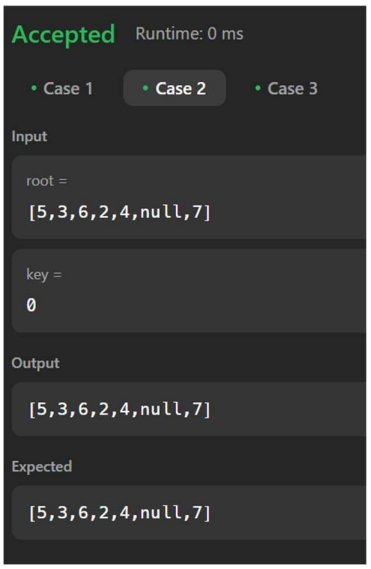

Accepted
Runtime: 0 ms
• Case 1
• Case 2
Input
root =
[1,2,3,4,null,5,6,null,null,7]
Output
7
Expected
7

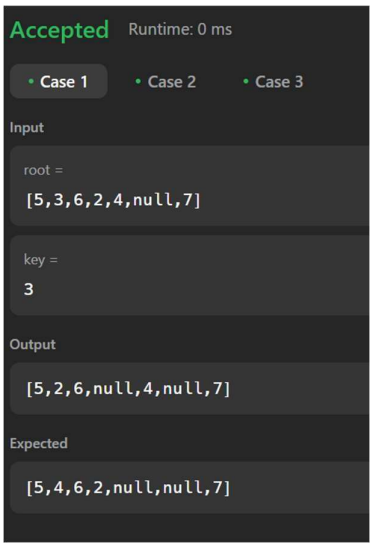
Leetcode problem 450: Delete Node in a BST

Code:

```
 Definition for a binary tree node.
 struct TreeNode {
     int val;
     struct TreeNode *left;
     struct TreeNode *right;
 };
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 :


Accepted
Runtime: 0 ms
• Case 1
• Case 2
• Case 3
Input
root =
[5,3,6,2,4,null,7]
key =
0
Output
[5,3,6,2,4,null,7]
Expected
[5,3,6,2,4,null,7]


Accepted
Runtime: 0 ms
• Case 1
• Case 2
• Case 3
Input
root =
[5,3,6,2,4,null,7]
key =
3
Output
[5,2,6,null,4,null,7]
Expected
[5,4,6,2,null,null,7]


Accepted
Runtime: 0 ms
• Case 1
• Case 2
• Case 3
Input
root =
[]
key =
0
Output
[]
Expected
[]