

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT On

DATA STRUCTURES (23CS3PCDST)

Submitted by

SUDARSHAN KOMAR(1BM22CS291)

**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 SUDARSHAN KOMAR (**1BM22CS291**), 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. 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	Working of stack using an array	4-6
2	WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression	7-9
3	WAP to simulate the working of a queue of integers using an array. WAP to simulate the working of a circular queue of integers using an array	10-18
4	WAP to Implement Singly Linked List (Insertion)	19-23
5	WAP to Implement Singly Linked List (Deletion)	24-30
6	WAP to Implement Single Link List (Sorting, Reversing and Concatenation). WAP to implement Stack & Queues using Linked Representation .	30-45
7	WAP to Implement doubly link list with primitive operations	45-52
8	Constructing Binary Search Tree(BST)	52-56
9	BFS and DFS	57-60
10	Hash Function	60-64
11	Leetcode problems	64-66

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 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.

```
#include <stdio.h>

#include <stdlib.h>

#define STACK_SIZE 5

void push(int st[], int *top) {
    int item;

    if (*top == STACK_SIZE - 1)
        printf("Stack overflow\n");
    else {
        printf("\nEnter an item: ");
        scanf("%d", &item);

        (*top)++;
        st[*top] = item;
    }
}
```

```

void pop(int st[], int *top) {

if (*top == -1)

printf("Stack underflow\n");

else {

printf("\n%d item was deleted", st[*top]);

(*top)--;

}

}

void display(int st[], int *top) {

int i;

if (*top == -1) {

printf("Stack is empty\n");

return;

}

for (i = 0; i <= *top; i++)

printf("%d\t", st[i]);

}

int main() {

int st[STACK_SIZE], top = -1, c;

while (1) {

printf("\n1. Push\n2. Pop\n3. Display\n");

printf("\nEnter your choice: ");

scanf("%d", &c);

switch (c) {

case 1:

```

```

push(st, &top);

break;

case 2:

pop(st, &top);

break;

case 3:

display(st, &top);

break;

default:

printf("\nInvalid choice!!!");

exit(0);

}

}

return 0;

}

```

Output:

```

C:\Users\bmsce\Desktop\18M22CS291\stack.exe
Enter the operation
1.PUSH
2.POP
3.DISPLAY
4.-1 to stop
1
Enter the value
2
PUSH() operation is successfull
Enter the operation
1.PUSH
2.POP
3.DISPLAY
4.-1 to stop
1
Enter the value
3
PUSH() operation is successfull
Enter the operation
1.PUSH
2.POP
3.DISPLAY
4.-1 to stop
1
Enter the value
4
PUSH() operation is successfull
Enter the operation
1.PUSH
2.POP
3.DISPLAY
4.-1 to stop
2
POP() operation is successfull
Enter the operation
1.PUSH
2.POP
3.DISPLAY
4.-1 to stop
-1
Operation completed

```

Lab program 2:

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

```
#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;

}

int 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);

return 0;

}
```

Output:

Lab program 3a:

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

- a) Insert
- b) Delete

c) Display

The program should print appropriate messages for queue empty and queue overflow conditions.

```
#include <stdio.h>

#include <stdbool.h>

#define MAX_SIZE 5

typedef struct {
    int queue[MAX_SIZE];
    int front, rear;
    int size;
} Queue;

void initQueue(Queue *q) {
    q->front = 0;
    q->rear = -1;
    q->size = 0;
}

bool isEmpty(Queue *q) {
    return q->size == 0;
}

bool isFull(Queue *q) {
    return q->size == MAX_SIZE;
```

```

}

void enqueue(Queue *q, int item) {
    if (isFull(q)) {
        printf("Queue Overflow! Cannot insert element.\n");
        return;
    }

    q->rear = (q->rear + 1) % MAX_SIZE;

    q->queue[q->rear] = item;

    q->size++;

    printf("Inserted %d into the queue.\n", item);
}

int dequeue(Queue *q) {
    if (isEmpty(q)) {
        printf("Queue Underflow! Cannot delete element.\n");
        return -1;
    }

    int item = q->queue[q->front];

    q->front++;

    q->size--;

    printf("Deleted %d from the queue.\n", item);

    return item;
}

void display(Queue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty.\n");
    }
}

```

```

return;
}

printf("Queue elements: ");

for (int i = q->front; i <= q->rear; i++) {

printf("%d ", q->queue[i]);

}

printf("\n");

}

int main() {

Queue q;

initQueue(&q);

int choice, item;

do {

printf("\n1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

if (isFull(&q)) {

printf("Queue Overflow. Cannot enqueue.\n");

} else {

printf("Enter element to enqueue: ");

scanf("%d", &item);

enqueue(&q, item);

}

}
}

```

```
break;

case 2:

dequeue(&q);

break;

case 3:

display(&q);

break;

case 4:

printf("Exiting...\n");

break;

default:

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

}

} while (choice != 4);

return 0;

}
```

Output:

```
C:\Users\bmsce\Desktop\1BM22CS29T\standardq.exe
Enter the operation:
1.enqueue
2.dequeue
3.display
4.-1 to stop
Enter the operator
2
queue underflow
Enter the operator
1
Enter the number:
2
successfully enqueued
Enter the operator
1
Enter the number:
3
successfully enqueued
Enter the operator
1
Enter the number:
4
successfully enqueued
Enter the operator
1
Enter the number:
5
successfully enqueued
Enter the operator
1
Enter the number:
6
successfully enqueued
Enter the operator
1
Enter the number:
7
queue overflow
Enter the operator
3
Elements are:
2
3
4
5
6
Enter the operator
```

Lab program 3b:

WAP to simulate the working of a circular queue of integers using an array. Provide the following operations.

a) Insert

b) Delete

c) Display

The program should print appropriate messages for queue empty and queue overflow conditions.

```
#include <stdio.h>

#include <stdbool.h>

#define MAX_SIZE 5

typedef struct {
    int queue[MAX_SIZE];
    int front, rear;
    int size;
} CircularQueue;

void initQueue(CircularQueue *cq) {
```

```

cq->front = 0;

cq->rear = -1;

cq->size = 0;
}

bool isEmpty(CircularQueue *cq) {

return cq->size == 0;

}

bool isFull(CircularQueue *cq) {

return cq->size == MAX_SIZE;

}

void enqueue(CircularQueue *cq, int item) {

if (isFull(cq)) {

printf("Queue Overflow! Cannot insert element.\n");

return;

}

cq->rear = (cq->rear + 1) % MAX_SIZE;

cq->queue[cq->rear] = item;

cq->size++;

printf("Inserted %d into the queue.\n", item);

}

int dequeue(CircularQueue *cq) {

if (isEmpty(cq)) {

printf("Queue Underflow! Cannot delete element.\n");

return -1;

}

```

```

int item = cq->queue[cq->front];

cq->front = (cq->front + 1) % MAX_SIZE;

cq->size--;

printf("Deleted %d from the queue.\n", item);

return item;
}

void display(CircularQueue *cq) {

if (isEmpty(cq)) {

printf("Queue is empty.\n");

return;

}

printf("Queue elements: ");

int i, count;

for (count = 0, i = cq->front; count < cq->size; count++, i = (i + 1) % MAX_SIZE) {

printf("%d ", cq->queue[i]);

}

printf("\n");

}

int main() {

CircularQueue cq;

initQueue(&cq);

int choice, item;

do {

printf("\n1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");

printf("Enter your choice: ");

```



```
scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter element to enqueue: ");

scanf("%d", &item);

enqueue(&cq, item);

break;

case 2:

dequeue(&cq);

break;

case 3:

display(&cq);

break;

case 4:

printf("Exiting...\n");

break;

default:

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

}

} while (choice != 4);

return 0;

}
```

Output:

```
"C:\C LAB PROGRAMS 243\circularq.exe"
Enter 1.Enqueue
2.Dequeue
3.Display
4.-1 to stop execution
Enter operator
2
Queue is empty/underflow
-1 is Dequeued
Enter operator
1
Enter no
2
Enter operator
1
Enter no
3
Enter operator
1
Enter no
4
Queue is full/overflow
Enter operator
3
Queue : 2      3
Enter operator
2
2 is Dequeued
Enter operator
-1
Process returned -1 (0xFFFFFFFF)   execution time : 26.744 s
Press any key to continue.
```

Lab program 4:

WAP to Implement Singly Linked List with following operations

- Create a linked list.
- Insertion of a node at first position, at any position and at end of list.
- Display the contents of the linked list.

```
#include <stdio.h>

#include <stdlib.h>

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

```

Node* createNode(int data) {

Node *newNode = (Node*)malloc(sizeof(Node));

if (newNode == NULL) {

printf("Memory allocation failed!\n");

exit(1);

}

newNode->data = data;

newNode->next = NULL;

return newNode;

}

Node* insertAtBeginning(Node *head, int data) {

Node *newNode = createNode(data);

newNode->next = head;

return newNode;

}

Node* insertAtPosition(Node *head, int data, int position) {

if (position < 1) {

printf("Invalid position!\n");

return head;

}

Node *newNode = createNode(data);

if (position == 1 || head == NULL) {

newNode->next = head;

return newNode;

}

```

```

Node *current = head;

int count = 1;

while (count < position - 1 && current != NULL) {

current = current->next;

count++;

}

if (current == NULL) {

printf("Position out of range!\n");

return head;

}

newNode->next = current->next;

current->next = newNode;

return head;

}

Node* insertAtEnd(Node *head, int data) {

Node *newNode = createNode(data);

if (head == NULL) {

return newNode;

}

Node *current = head;

while (current->next != NULL) {

current = current->next;

}

current->next = newNode;

return head;

```

```

}

void displayList(Node *head) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }

    Node *current = head;

    printf("List elements: ");

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

        current = current->next;
    }

    printf("\n");
}

void freeList(Node *head) {
    Node *current = head;

    Node *temp;

    while (current != NULL) {
        temp = current;

        current = current->next;

        free(temp);
    }
}

int main() {
    Node *head = NULL;

```

```

int choice, data, position;

do {

printf("\n1. Insert at beginning\n2. Insert at position\n3. Insert at end\n4. Display\n5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data to insert at beginning: ");

scanf("%d", &data);

head = insertAtBeginning(head, data);

break;

case 2:

printf("Enter data to insert: ");

scanf("%d", &data);

printf("Enter position to insert at: ");

scanf("%d", &position);

head = insertAtPosition(head, data, position);

break;

case 3:

printf("Enter data to insert at end: ");

scanf("%d", &data);

head = insertAtEnd(head, data);

break;

case 4:

displayList(head);

```

```

break;

case 5:

freeList(head);

printf("Exiting...\n");

break;

default:

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

}

} while (choice != 5);

return 0;

}

```

Output:

```

C:\Users\bmsce\Desktop\1BM22CS291\lins.exe
Enter 1. insert at the end
2. insert at the start
3. insert before a node
4. insert after a node
5. -1 to stop
Enter operation 1
Enter the element to insert at the end
5
Elements are: 5
Enter operation 2
Enter the element to insert at the start
4
Elements are: 4 5
Enter operation 3
Enter the element to insert
3
Enter the data of the node before which to insert
3
Node with data 3 not found. Cannot insert before the node.
Elements are: 4 5
Enter operation 3
Enter the element to insert
4
Enter the data of the node after which to insert
4
Elements are: 3 4 8 5
Enter operation -1
Execution stopped

Process returned 0 (0x0) execution time : 66.235 s
Press any key to continue.

```

Lab program 5:

WAP to Implement Singly Linked List with following operations

- a) Create a linked list.
- b) Deletion of first element, specified element and last element in the list.
- c) Display the contents of the linked list.

```
#include <stdio.h>

#include <stdlib.h>

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

Node* createNode(int data) {
    Node *newNode = (Node*)malloc(sizeof(Node));

    if (newNode == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }

    newNode->data = data;
    newNode->next = NULL;

    return newNode;
}
```



```

Node* insertAtBeginning(Node *head, int data) {

Node *newNode = createNode(data);

newNode->next = head;

return newNode;

}

Node* deleteFirstNode(Node *head) {

if (head == NULL) {

printf("List is empty. Nothing to delete.\n");

return NULL;

}

Node *temp = head;

head = head->next;

free(temp);

printf("Deleted the first node from the list.\n");

return head;

}

Node* deleteSpecifiedNode(Node *head, int key) {

Node *current = head;

Node *prev = NULL;

if (current != NULL && current->data == key) {

head = head->next;

free(current);

printf("Deleted node with key %d from the list.\n", key);

return head;

}

```

```

while (current != NULL && current->data != key) {

prev = current;

current = current->next;

}

if (current == NULL) {

printf("Key %d not found in the list.\n", key);

return head;

}

prev->next = current->next;

free(current);

printf("Deleted node with key %d from the list.\n", key);

return head;

}

Node* deleteLastNode(Node *head) {

if (head == NULL) {

printf("List is empty. Nothing to delete.\n");

return NULL;

}

if (head->next == NULL) {

free(head);

printf("Deleted the last node from the list.\n");

return NULL;

}

Node *prev = NULL;

Node *current = head;

```

```

while (current->next != NULL) {

prev = current;

current = current->next;

}

prev->next = NULL;

free(current);

printf("Deleted the last node from the list.\n");

return head;

}

void displayList(Node *head) {

if (head == NULL) {

printf("List is empty.\n");

return;

}

Node *current = head;

printf("List elements: ");

while (current != NULL) {

printf("%d ", current->data);

current = current->next;

}

printf("\n");

}

void freeList(Node *head) {

Node *current = head;

Node *temp;

```

```

while (current != NULL) {

temp = current;

current = current->next;

free(temp);

}

}

int main() {

Node *head = NULL;

int choice, data, key;

do {

printf("\n1. Insert at beginning\n2. Delete first node\n3. Delete specified node\n4. Delete last\n5. Display\n6. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data to insert at beginning: ");

scanf("%d", &data);

head = insertAtBeginning(head, data);

break;

case 2:

head = deleteFirstNode(head);

break;

case 3:

printf("Enter the key of node to delete: ");

```

```
scanf("%d", &key);

head = deleteSpecifiedNode(head, key);

break;

case 4:

head = deleteLastNode(head);

break;

case 5:

displayList(head);

break;

case 6:

freeList(head);

printf("Exiting...\n");

break;

default:

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

}

} while (choice != 6);

return 0;

}
```

Output:

```
C:\Users\bmsce\Desktop\1BM22CS291\lidel.exe
Enter 1. insert
2. delete at the start
3. delete at the end
4. delete before a node
5. delete after a node
6. -1 to stop
Enter operation 1
Enter the element to insert
3
Elements are: 3
Enter operation 1
Enter the element to insert
4
Elements are: 3 4
Enter operation 1
Enter the element to insert
5
Elements are: 3 4 5
Enter operation 2
Deleted node: 3
Elements are: 4 5
Enter operation 3
Deleted node: 5
Elements are: 4
Enter operation 1
Enter the element to insert
5
Elements are: 4 5
Enter operation 4
Enter the data of the node before which to delete
5
Enter 1. insert
2. delete at the start
3. delete at the end
4. delete before a node
5. delete after a node
6. -1 to stop
Deleted node: 5
Elements are: 4
Enter operation
```

Lab program 6a:

WAP to Implement Single Link List with following operations

- a) Sort the linked list.
- b) Reverse the linked list.
- c) Concatenation of two linked lists

```
#include <stdio.h>

#include <stdlib.h>

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

```

Node* createNode(int data) {

Node *newNode = (Node*)malloc(sizeof(Node));

if (newNode == NULL) {

printf("Memory allocation failed!\n");

exit(1);

}

newNode->data = data;

newNode->next = NULL;

return newNode;

}

Node* insertAtBeginning(Node *head, int data) {

Node *newNode = createNode(data);

newNode->next = head;

return newNode;

}

void displayList(Node *head) {

if (head == NULL) {

printf("List is empty.\n");

return;

}

Node *current = head;

printf("List elements: ");

while (current != NULL) {

printf("%d ", current->data);

current = current->next;

```

```

}

printf("\n");

}

Node* sortLinkedList(Node *head) {
    if (head == NULL || head->next == NULL)
        return head;

    Node *prev = head;
    Node *current = head->next;

    while (current != NULL) {
        Node *innerPrev = NULL;
        Node *innerCurrent = head;

        while (innerCurrent != current) {
            if (innerCurrent->data > current->data) {
                prev->next = current->next;
                current->next = innerCurrent;

                if (innerPrev == NULL)
                    head = current;
                else
                    innerPrev->next = current;

                current = prev->next;
                break;
            }

            innerPrev = innerCurrent;
            innerCurrent = innerCurrent->next;
        }
    }
}

```



```

if (innerCurrent == current) {

prev = current;

current = current->next;

}

}

return head;

}

Node* reverseLinkedList(Node *head) {

Node *prev = NULL;

Node *current = head;

Node *next = NULL;

while (current != NULL) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

head = prev;

return head;

}

Node* concatenateLinkedLists(Node *list1, Node *list2) {

if (list1 == NULL)

return list2;

if (list2 == NULL)

return list1;

```

```
Node *current = list1;

while (current->next != NULL) {

current = current->next;

}

current->next = list2;

return list1;

}

int main() {

Node *list1 = NULL;

Node *list2 = NULL;

list1 = insertAtBeginning(list1, 30);

list1 = insertAtBeginning(list1, 20);

list1 = insertAtBeginning(list1, 10);
```

```
printf("List 1:\n");

displayList(list1);

list1 = sortLinkedList(list1);

printf("Sorted List 1:\n");

displayList(list1);

list1 = reverseLinkedList(list1);

printf("Reversed List 1:\n");

displayList(list1);

list2 = insertAtBeginning(list2, 60);

list2 = insertAtBeginning(list2, 50);

list2 = insertAtBeginning(list2, 40);
```

```

printf("List 2:\n");

displayList(list2);

list2 = sortLinkedList(list2);

printf("Sorted List 2:\n");

displayList(list2);

list2 = reverseLinkedList(list2);

printf("Reversed List 2:\n");

displayList(list2);

```

```

Node *concatenatedList = concatenateLinkedLists(list1, list2);

printf("Concatenated List:\n");

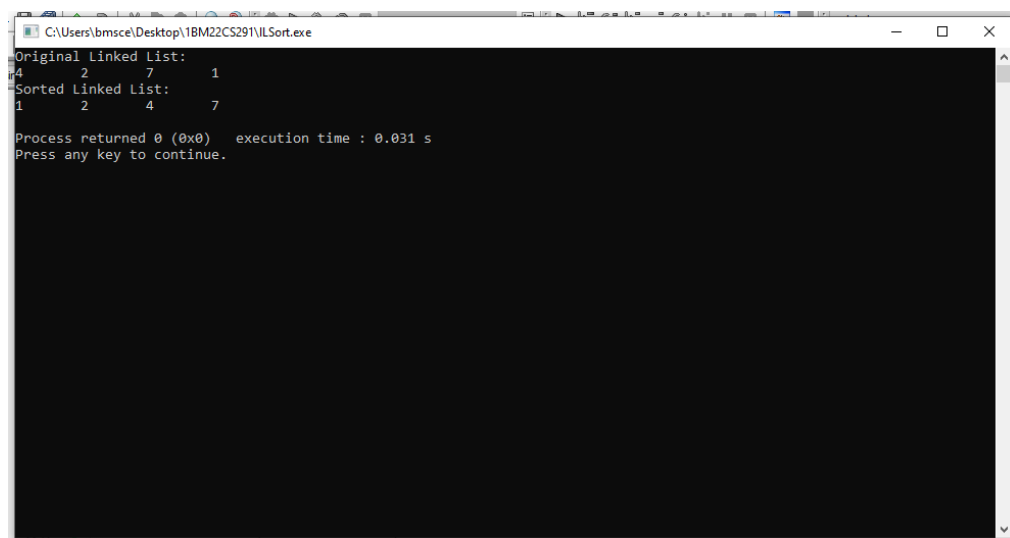
displayList(concatenatedList);

return 0;

}

```

Output:



```

C:\Users\bmsce\Desktop\1BM22CS291\VLSort.exe
Original linked list:
4 2 7 1
Sorted linked list:
1 2 4 7
Reversed linked list:
1 2 4 7
Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.

```

```
C:\Users\bmsce\Desktop\1BM22CS291\VLRev.exe
enter the number of elements:
4
Enter the element 1:
1
Enter the element 2:
2
Enter the element 3:
3
Enter the element 4:
4
Original Linked List:
1 2 3 4
Reversed Linked List:
4 3 2 1
Process returned 0 (0x0) execution time : 32.735 s
Press any key to continue.

C:\Users\bmsce\Desktop\1BM22CS291\VLConc.exe
Original Linked List 1:
10
Original Linked List 2:
20
Concatenated Linked List:
10 20
Process returned 0 (0x0) execution time : 0.016 s
Press any key to continue.
```

Lab program 6b:

WAP to implement Stack & Queues using Linked Representation

```
#include <stdio.h>

#include <stdlib.h>

typedef struct StackNode {
    int data;
    struct StackNode* next;
};
```

```

} StackNode;

StackNode* createStackNode(int data) {

StackNode* newNode = (StackNode*)malloc(sizeof(StackNode));

if (newNode == NULL) {

printf("Memory allocation failed!\n");

exit(1);

}

newNode->data = data;

newNode->next = NULL;

return newNode;

}

int isEmpty(StackNode* root) {

return (root == NULL);

}

void push(StackNode** root, int data) {

StackNode* newNode = createStackNode(data);

newNode->next = *root;

*root = newNode;

printf("Pushed %d onto the stack.\n", data);

}

int pop(StackNode** root) {

if (isEmpty(*root)) {

printf("Stack Underflow! Cannot pop element.\n");

return -1;

}

```

```
int popped = (*root)->data;

StackNode* temp = *root;

*root = (*root)->next;

free(temp);

return popped;
}

int peek(StackNode* root) {

if (isEmpty(root)) {

printf("Stack is empty.\n");

return -1;

}

return root->data;

}

void displayStack(StackNode* root) {

if (isEmpty(root)) {

printf("Stack is empty.\n");

return;

}

printf("Stack elements: ");

while (root != NULL) {

printf("%d ", root->data);

root = root->next;

}

printf("\n");

}
```

```

int main() {

StackNode* stack = NULL;

int choice, data;

do {

printf("\n1. Push\n2. Pop\n3. Peek\n4. Display\n5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data to push onto the stack: ");

scanf("%d", &data);

push(&stack, data);

break;

case 2:

printf("Popped %d from the stack.\n", pop(&stack));

break;

case 3:

printf("Top element of the stack: %d\n", peek(stack));

break;

case 4:

displayStack(stack);

break;

case 5:

printf("Exiting...\n");

break;

```

```

default:

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

}

} while (choice != 5);

return 0;

}

```

Output:

```

C:\Users\bmsce\Desktop\18M22CS291\stackUsingLL.exe
Enter 1. Push
2. Pop
3. -1 to stop
Enter operation:
2
Stack Underflow
Stack is empty
Enter operation:
1
Enter the element to push
2
Stack elements are: 2
Enter operation:
1
Enter the element to push
3
Stack elements are: 3 2
Enter operation:
1
Enter the element to push
4
Stack elements are: 4 3 2
Enter operation:
2
Popped element:4
Stack elements are: 3 2
Enter operation:
2
Popped element:3
Stack elements are: 2
Enter operation:
-1
Execution stopped
Process returned 0 (0x0) execution time : 15.516 s
Press any key to continue.

```

Queue:

```

#include <stdio.h>

#include <stdlib.h>

typedef struct QueueNode {

```



```

int data;

struct QueueNode* next;
} QueueNode;

typedef struct {
    QueueNode* front;
    QueueNode* rear;
} Queue;

QueueNode* createQueueNode(int data) {
    QueueNode* newNode = (QueueNode*)malloc(sizeof(QueueNode));

    if (newNode == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }

    newNode->data = data;
    newNode->next = NULL;

    return newNode;
}

Queue* createQueue() {
    Queue* queue = (Queue*)malloc(sizeof(Queue));

    if (queue == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }

    queue->front = queue->rear = NULL;

    return queue;
}

```

```

}

int isEmpty(Queue* queue) {
    return (queue->front == NULL);
}

void enqueue(Queue* queue, int data) {
    QueueNode* newNode = createQueueNode(data);

    if (isEmpty(queue)) {
        queue->front = queue->rear = newNode;
    } else {
        queue->rear->next = newNode;
        queue->rear = newNode;
    }

    printf("Enqueued %d into the queue.\n", data);
}

int dequeue(Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue Underflow! Cannot dequeue element.\n");
        return -1;
    }

    int dequeued = queue->front->data;
    QueueNode* temp = queue->front;
    queue->front = queue->front->next;

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

```

```

free(temp);

return dequeued;
}

int peek(Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty.\n");
        return -1;
    }

    return queue->front->data;
}

void displayQueue(Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty.\n");
        return;
    }

    printf("Queue elements: ");

    QueueNode* current = queue->front;

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

        current = current->next;
    }

    printf("\n");
}

int main() {

    Queue* queue = createQueue();

```

```
int choice, data;

do {

printf("\n1. Enqueue\n2. Dequeue\n3. Peek\n4. Display\n5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data to enqueue into the queue: ");

scanf("%d", &data);

enqueue(queue, data);

break;

case 2:

printf("Dequeued %d from the queue.\n", dequeue(queue));

break;

case 3:

printf("Front element of the queue: %d\n", peek(queue));

break;

case 4:

displayQueue(queue);

break;

case 5:

printf("Exiting...\n");

break;

default:

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

```
}  
  
} while (choice != 5);  
  
return 0;  
  
}
```

Output:

```

C:\Users\bmsce\Desktop\IBM22CS291\queueUsingLL.exe
Enter 1. Enqueue
2. Dequeue
3. -1 to stop
Enter operation:
2
Queue Underflow
Queue is empty
Enter operation:
-1
Enter the element to enqueue
2
Queue elements are: 2
Enter operation:
1
Enter the element to enqueue
3
Queue elements are: 2 3
Enter operation:
1
Enter the element to enqueue
5
Queue elements are: 2 3 5
Enter operation:
2
Dequeued Element: 2
Queue elements are: 3 5
Enter operation:
2
Dequeued Element: 3
Queue elements are: 5
Enter operation:
-1
Execution stopped

Process returned 0 (0x0) execution time : 11.357 s
Press any key to continue.

```

Lab program 7:

WAP to Implement doubly link list with primitive operations

- a) Create a doubly linked list.
- b) Insert a new node to the left of the node.
- c) Delete the node based on a specific value
- d) Display the contents of the list

```

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {
    int data;

    struct Node* prev;

    struct Node* next;
} Node;

Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));

    if (newNode == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }

    newNode->data = data;

    newNode->prev = NULL;

    newNode->next = NULL;

    return newNode;
}

void insertLeft(Node** head, Node* node, int data) {
    Node* newNode = createNode(data);

    newNode->next = node;

    newNode->prev = node->prev;

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

```

```

} else {

*head = newNode;

}

node->prev = newNode;

}

void deleteNode(Node** head, int key) {

Node* current = *head;

while (current != NULL) {

if (current->data == key) {

if (current->prev != NULL) {

current->prev->next = current->next;

} else {

*head = current->next;

}

if (current->next != NULL) {

current->next->prev = current->prev;

}

free(current);

return;

}

current = current->next;

}

printf("Node with value %d not found in the list.\n", key);

}

```

```

void displayList(Node* head) {

if (head == NULL) {

printf("List is empty.\n");

return;

}

printf("List elements: ");

while (head != NULL) {

printf("%d ", head->data);

head = head->next;

}

printf("\n");

}

void freeList(Node* head) {

Node* current = head;

Node* temp;

while (current != NULL) {

temp = current;

current = current->next;

free(temp);

}

}

int main() {

Node* head = NULL;

int choice, data, value;

do {

```



```

printf("\n1. Create a Doubly Linked List\n2. Insert a new node to the left of a node\n3. Delete
a node based on a specific value\n4. Display the contents of the list\n5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the number of elements to create the list: ");

scanf("%d", &data);

printf("Enter the elements: ");

for (int i = 0; i < data; ++i) {

int value;

scanf("%d", &value);

if (head == NULL) {

head = createNode(value);

} else {

Node* temp = head;

while (temp->next != NULL) {

temp = temp->next;

}

Node* newNode = createNode(value);

temp->next = newNode;

newNode->prev = temp;

}

}

break;

```

```

case 2:

if (head == NULL) {

printf("List is empty. Create a list first.\n");

break;

}

printf("Enter the value of the node to the left of which you want to insert a new node: ");

scanf("%d", &value);

printf("Enter the data of the new node: ");

scanf("%d", &data);

Node* current = head;

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

current = current->next;

}

if (current == NULL) {

printf("Node with value %d not found in the list.\n", value);

} else {

insertLeft(&head, current, data);

}

break;

case 3:

if (head == NULL) {

printf("List is empty. Create a list first.\n");

break;

}

printf("Enter the value of the node you want to delete: ");

```

```
scanf("%d", &data);

deleteNode(&head, data);

break;

case 4:

displayList(head);

break;

case 5:

printf("Exiting...\n");

break;

default:

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

}

} while (choice != 5);

freeList(head);

return 0;

}
```

Output:

```
C:\Users\bmsce\Desktop\18M22CS291\doubleLinkedList.exe
1
Enter the number of nodes to create initially: 3
Enter data for node 1: 2
Enter data for node 2: 3
Enter data for node 3: 4
1.Insert
2.Delete
3.toStopDoubly Linked List: 2 3 4
Enter your choice: 1
Enter the data of the target node: 2
Enter data for the new node: 1
Node with data 1 inserted to the left of node with data 2.
Doubly Linked List: 1 2 3 4
Enter your choice: 2
Enter the value to delete: 4
Node with data 4 deleted.
Doubly Linked List: 1 2 3
Enter your choice: 2
Enter the value to delete: 3
Node with data 3 deleted.
Doubly Linked List: 1 2
Enter your choice: 3
Process returned 0 (0x0) execution time : 25.026 s
Press any key to continue.
```

Lab program 8:

Write a program

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

```
#include <stdio.h>

#include <stdlib.h>

typedef struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
} TreeNode;
```

```

TreeNode* createNode(int data) {

TreeNode* newNode = (TreeNode*)malloc(sizeof(TreeNode));

if (newNode == NULL) {

printf("Memory allocation failed!\n");

exit(1);

}

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

TreeNode* insertNode(TreeNode* root, int data) {

if (root == NULL) {

return createNode(data);

}

if (data < root->data) {

root->left = insertNode(root->left, data);

} else if (data > root->data) {

root->right = insertNode(root->right, data);

}

return root;

}

void inorderTraversal(TreeNode* root) {

if (root != NULL) {

inorderTraversal(root->left);

```

```

printf("%d ", root->data);

inorderTraversal(root->right);

}

}

void preorderTraversal(TreeNode* root) {

if (root != NULL) {

printf("%d ", root->data);

preorderTraversal(root->left);

preorderTraversal(root->right);

}

}

void postorderTraversal(TreeNode* root) {

if (root != NULL) {

postorderTraversal(root->left);

postorderTraversal(root->right);

printf("%d ", root->data);

}

}

void displayTree(TreeNode* root) {

printf("Elements in the tree (inorder traversal): ");

inorderTraversal(root);

printf("\n");

}

int main() {

TreeNode* root = NULL;

```

```

int choice, data;

do {

printf("\n1. Insert\n2. Inorder Traversal\n3. Preorder Traversal\n4. Postorder Traversal\n5.
Display Tree\n6. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data to insert into the tree: ");

scanf("%d", &data);

root = insertNode(root, data);

break;

case 2:

printf("Inorder Traversal: ");

inorderTraversal(root);

printf("\n");

break;

case 3:

printf("Preorder Traversal: ");

preorderTraversal(root);

printf("\n");

break;

case 4:

printf("Postorder Traversal: ");

postorderTraversal(root);

```

```

printf("\n");

break;

case 5:

displayTree(root);

break;

case 6:

printf("Exiting...\n");

break;

default:

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

}

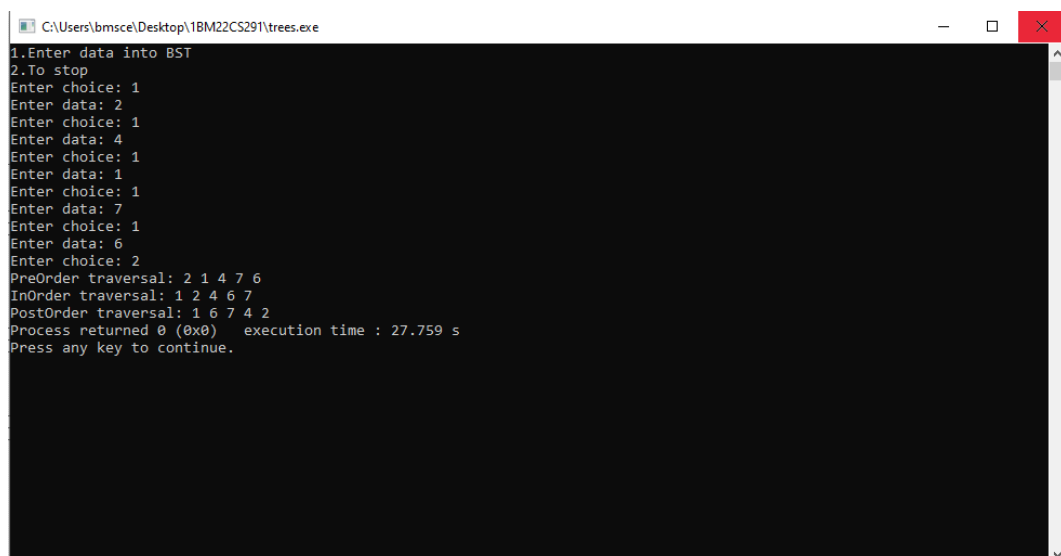
} while (choice != 6);

return 0;

}

```

Output:



```

C:\Users\bmsce\Desktop\1BM22CS291\trees.exe
1.Enter data into BST
2.To stop
Enter choice: 1
Enter data: 2
Enter choice: 1
Enter data: 4
Enter choice: 1
Enter data: 1
Enter choice: 1
Enter data: 7
Enter choice: 1
Enter data: 6
Enter choice: 2
PreOrder traversal: 2 1 4 7 6
InOrder traversal: 1 2 4 6 7
PostOrder traversal: 1 6 7 4 2
Process returned 0 (0x0)   execution time : 27.759 s
Press any key to continue.

```


Lab program 9:

Write a Program to traverse a graph using BFS method.

```
#include <stdio.h>

void bfs(int a[10][10], int n, int u) {
    int f = 0, r = -1, q[10] = {0}, v, s[10] = {0};
    printf("The nodes visited from %d: ", u);
    q[++r] = u;
    s[u] = 1;
    printf("%d ", u);
    while (f <= r) {
        u = q[f++];
        for (v = 0; v < n; v++) {
            if (a[u][v] == 1 && s[v] == 0) {
                printf("%d ", v);
                s[v] = 1;
                q[++r] = v;
            }
        }
    }
    printf("\n");
}

int main() {
```

```

int n, a[10][10], source, i, j;

printf("\nEnter the number of nodes: ");

scanf("%d", &n);

printf("\nEnter the adjacency matrix:\n");

for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        scanf("%d", &a[i][j]);
    }
}

for (source = 0; source < n; source++) {
    bfs(a, n, source);
}

return 0;
}

```

Output:

```

Enter the number of nodes: 4

Enter the adjacency matrix:
0 1 1 0
1 0 1 1
1 1 0 1
0 1 1 0
The nodes visited from 0: 0 1 2 3
The nodes visited from 1: 1 0 2 3
The nodes visited from 2: 2 0 1 3
The nodes visited from 3: 3 1 2 0

```

b)Write a program to check wheater given graph is connected or not using DFS method

```

#include <stdio.h>

#include <stdbool.h>

#define MAX_SIZE 100

int n;

int a[MAX_SIZE][MAX_SIZE];

int s[MAX_SIZE];

void dfs(int v) {

    s[v] = 1;

    for (int i = 1; i <= n; i++) {

        if (a[v][i] && !s[i]) {

            dfs(i);

        }

    }

}

int main() {

    int i, j, count = 0;

    printf("\nEnter number of vertices: ");

    scanf("%d", &n);

    for (i = 1; i <= n; i++) {

        s[i] = 0;

        for (j = 1; j <= n; j++) {

            a[i][j] = 0;

        }

    }

}

```

```

printf("Enter the adjacency matrix:\n");

for (i = 1; i <= n; i++) {
    for (j = 1; j <= n; j++) {
        scanf("%d", &a[i][j]);
    }
}

dfs(1);

for (i = 1; i <= n; i++) {
    if (s[i]) {
        count++;
    }
}

if (count == n) {
    printf("Graph is connected\n");
} else {
    printf("Graph is not connected\n");
}

return 0;
}

```

Output:

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

Lab Program 10:

Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are integers.

Design and develop a Program in C that uses Hash function H: $K \rightarrow L$ as $H(K) = K \bmod m$ (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.

```
#include <stdio.h>

#include <stdlib.h>

#define MAX_EMPLOYEES 100

#define HT_SIZE 10

typedef struct {
    int key;
} Employee;

typedef struct {
    Employee* entries[HT_SIZE];
} HashTable;

int hash(int key) {
    return key % HT_SIZE;
}
```

```

void initHashTable(HashTable* ht) {

for (int i = 0; i < HT_SIZE; i++) {

ht->entries[i] = NULL;

}

}

void insertEmployee(HashTable* ht, Employee* emp) {

int index = hash(emp->key);

while (ht->entries[index] != NULL) {

index = (index + 1) % HT_SIZE;

}

ht->entries[index] = emp;

}

void displayHashTable(HashTable* ht) {

printf("\nHash Table:\n");

for (int i = 0; i < HT_SIZE; i++) {

if (ht->entries[i] != NULL) {

printf("Index %d: Key %d\n", i, ht->entries[i]->key);

} else {

printf("Index %d: Empty\n", i);

}

}

}

int main() {

HashTable ht;

initHashTable(&ht);

```

```
int n;

printf("Enter the number of employee records: ");

scanf("%d", &n);

printf("Enter the employee keys:\n");

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

    Employee* emp = (Employee*)malloc(sizeof(Employee));

    if (emp == NULL) {

        printf("Memory allocation failed!\n");

        exit(1);

    }

    scanf("%d", &emp->key);

    insertEmployee(&ht, emp);

}

displayHashTable(&ht);

return 0;

}
```

Output:

```

Enter the number of employee records: 6
Enter the employee keys:
12
22
45
67
68
46

Hash Table:
Index 0: Empty
Index 1: Empty
Index 2: Key 12
Index 3: Key 22
Index 4: Empty
Index 5: Key 45
Index 6: Key 46
Index 7: Key 67
Index 8: Key 68
Index 9: Empty

```

Leet Code Problem:

ScoreOfParentheses:

The screenshot displays the LeetCode submission interface for the "ScoreOfParentheses" problem. The submission is marked as "Accepted" and was submitted by "komar_sudarshan" on Feb 19, 2024, at 10:30. The performance metrics show a runtime of 2 ms (beating 69.44% of users with C) and a memory usage of 5.74 MB (beating 27.78% of users with C). The C code is shown in the editor, and the test results for Case 1 are displayed at the bottom.

Runtime: 2 ms
Beats 69.44% of users with C

Memory: 5.74 MB
Beats 27.78% of users with C

```

1 int scoreOfParentheses(char* s) {
2     int* st = (int*)malloc(strlen(s) * sizeof(int));
3     int top = -1;
4     int score = 0;
5     for (int i = 0; i < strlen(s); i++) {
6         if (s[i] == '(') {
7             st[++top] = score;
8             score = 0;
9         } else {
10            score = st[top--] + ((2 * score) > 1 ? (2 * score) : 1);
11        }
12    }
13    free(st);
14    return score;
15 }

```

Testcase 1: Case 1 Case 2 Case 3 +

Input: s = "()"

Output: 1

Delete the Middle Node Of a Linked List:

The screenshot shows a submission for the problem "Delete the Middle Node Of a Linked List". The submission is accepted, with a runtime of 333 ms and memory usage of 77.86 MB. The code is written in C++ and implements a function to delete the middle node of a singly-linked list. The code is as follows:

```
6  *  
7  */  
8  struct ListNode* deleteMiddle(struct ListNode* head) {  
9      int count=0;  
10     struct ListNode *ptr;  
11     ptr=head;  
12     while(ptr!=NULL){  
13         count++;  
14         ptr=ptr->next;  
15     }  
16     int mid=count/2;  
17     struct ListNode *prev=NULL;  
18     ptr=head;  
19     if(head->next==NULL){  
20         head=NULL;  
21         return head;  
22     }  
23     for(int i=0;i<mid;i++){  
24         prev=ptr;  
25         ptr=ptr->next;  
26     }  
27     prev->next=ptr->next;  
28     free(ptr);  
29     return head;  
30 }
```

The code is saved to local storage. The test result shows that the solution is accepted for all test cases.

Odd Even Linked List

The screenshot shows a submission for the problem "Odd Even Linked List". The submission is accepted, with a runtime of 6 ms and memory usage of 6.67 MB. The code is written in C++ and implements a function to rearrange a singly-linked list into an odd-even linked list. The code is as follows:

```
20 // prev->next=temp;  
21 // return head;  
22 // }  
23 struct ListNode* oddEvenList(struct ListNode* head) {  
24     if (head == NULL || head->next == NULL) {  
25         return head;  
26     }  
27     struct ListNode* oddHead = head;  
28     struct ListNode* evenHead = head->next;  
29     struct ListNode* odd = oddHead;  
30     struct ListNode* even = evenHead;  
31     while (even!=NULL && even->next!=NULL) {  
32         odd->next = even->next;  
33         odd = odd->next;  
34         even->next = odd->next;  
35         even = even->next;  
36     }  
37     odd->next = evenHead;  
38     return head;  
39 }  
40
```

The code is saved to local storage. The test result shows that the solution is accepted for all test cases.

Delete Node In BST

The screenshot shows a LeetCode submission for the problem "Delete Node In BST". The submission is accepted, with a runtime of 18 ms and memory usage of 13.93 MB. The code is written in C and implements a recursive function to delete a node from a Binary Search Tree (BST).

Runtime: 18 ms
Beats 58.88% of users with C

Memory: 13.93 MB
Beats 24.56% of users with C

```
9 struct TreeNode* deleteNode(struct TreeNode* root, int key) {
10     if (root == NULL) {
11         return root;
12     }
13     if (key < root->val) {
14         root->left = deleteNode(root->left, key);
15     }
16     else if (key > root->val) {
17         root->right = deleteNode(root->right, key);
18     }
19     else {
20         if (root->left == NULL) {
21             struct TreeNode* temp = root->right;
22             free(root);
23             return temp;
24         }
25         else if (root->right == NULL) {
26             struct TreeNode* temp = root->left;
27             free(root);
28             return temp;
29         }
30         struct TreeNode* temp = root->right;
```

Find Bottom Left Tree Value

The screenshot shows a LeetCode submission for the problem "Find Bottom Left Tree Value". The submission is accepted, with a runtime of 4 ms and memory usage of 10.48 MB. The code is written in C and implements a breadth-first search (BFS) algorithm to find the value of the bottom-left node in a binary tree.

Runtime: 4 ms
Beats 83.51% of users with C

Memory: 10.48 MB
Beats 8.65% of users with C

```
9 int findBottomLeftValue(struct TreeNode* root) {
10     if (root == NULL) {
11         return -1;
12     }
13     struct TreeNode* queue[10000];
14     int front = 0, rear = 0;
15     int leftmostValue = -1;
16     queue[rear++] = root;
17     while (front < rear) {
18         int levelSize = rear - front;
19         for (int i = 0; i < levelSize; ++i) {
20             struct TreeNode* node = queue[front++];
21             if (i == 0) {
22                 leftmostValue = node->val;
23             }
24             if (node->left != NULL) {
25                 queue[rear++] = node->left;
26             }
27             if (node->right != NULL) {
28                 queue[rear++] = node->right;
29             }
30         }
31     }
```