

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT On

DATA STRUCTURES (23CS3PCDST)

Submitted by

Nagaraja G

(24BECS426)

**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
September 2024-January 2025**

**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 **Nagaraja G (24BECS426)**, 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 2024-25. 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. Namratha M
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Kavitha Sooda
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Sl. No.	Date	Experiment Title	Page No.
1	09/10/2024	Program to simulate the working of Stack	4-6
2	09/10/2024	Program to convert a given valid parenthesized Infix arithmetic expression to Postfix expression	6-8
3	16/10/2024 23/10/2024	Program to simulate the working of a Queue LeetCode: Queue using Stack Program to simulate the working of a Circular queue LeetCode: Valid Parentheses	8-19
4	30/10/2024	Program to Implement Singly Linked List (Create, Insert, Display)	19-24
5	30/10/2024	Program to Implement Singly Linked List (Create, Delete, Display)	25-31
6	13/11/2024	Linked list operations (Sort, Reverse, Concatenate). Implementation of Queue and Stack using Linked List.	31-40
7	20/11/2024	Program to Implement doubly link list. LeetCode: Finding middle element of a linked list.	41-46
8	27/11/2024	Program to Implement Binary Search Tree LeetCode: Intersection of Two Linked list	46-50
9	04/12/2024	BFS traversal and DFS Traversal	51-53
10	17/12/2024	Program to implement Hashtable	54-56

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>
#define N 5

int stack[N];
int top=-1;

void push(){
int x;
printf("Enter data:\n");
scanf("%d",&x);
if (top==N-1){
    printf("Stack Overflow occurred!");
}
else {
    top++;
    stack[top]=x;
}
}

void pop(){
int item;
if(top== -1){
    printf("Stack underflow occurred!");
}
else{
    item=stack[top];
    top--;
}
printf("Deleted item is:%d\n",item);
}

void peek(){
if(top== -1)
{
    printf("Stack is Empty");
}
else{
    printf("Top element is :%d\n",stack[top]);
}
}
```

```

    }
}

void display(){
int i;
for (i=top;i>=0;i--){
    printf("%d\t",stack[i]);
}
}

void main(){
int ch;
do{
    printf("Enter choice for operation b/w 1 to 4 :");
    scanf("%d",&ch);
switch(ch){
case 1: push();
        break;
case 2: pop();
        break;
case 3: peek();
        break;
case 4: display();
        break;
default :
        printf("Invalid choice !");
        }
}while(ch!=0);

}

```

Output:

C:\Users\STUDENT\Desktop\347\stack\bin\Debug\stack.exe

```
Enter choice for operation b/w 1 to 41
Enter data:
1
Enter choice for operation b/w 1 to 41
Enter data:
2
Enter choice for operation b/w 1 to 41
Enter data:
3
Enter choice for operation b/w 1 to 41
Enter data:
4
Enter choice for operation b/w 1 to 41
Enter data:
5
Enter choice for operation b/w 1 to 41
Enter data:
6
Stack Overflow occurred!Enter choice for operation b/w 1 to 44
5      4      3      2      1      Enter choice for operation b/w 1 to 43
Top element is :5
Enter choice for operation b/w 1 to 42
Deleted item is:5
Enter choice for operation b/w 1 to 44
4      3      2      1      Enter choice for operation b/w 1 to 42
Deleted item is:4
Enter choice for operation b/w 1 to 4
2
Deleted item is:3
Enter choice for operation b/w 1 to 42
Deleted item is:2
Enter choice for operation b/w 1 to 42
Deleted item is:1
Enter choice for operation b/w 1 to 42
Stack underflow occurred!Deleted item is:0
Enter choice for operation b/w 1 to 4
```

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 <stdlib.h>
#include <ctype.h>
#define max 100
```

```
int top = -1;
char stack[max];
```

```
void push(char item)
{
```

```

    stack[++(top)]=item;
}
char pop()
{
    return stack[(top)--];
}
int prec(char item)
{
    switch(item)
    {
        case '(':
            return 1;
        case '+':
        case '-':
            return 2;
        case '*':
        case '/':
            return 3;
        default:
            return -1;
    }
}
void infixtopostfix(char *infix)
{
    char postfix[max];
    int i = 0;
    for(int k=0;infix[k]!='\0';k++)
    {
        char item=infix[k];
        if (isalnum(item))
        {
            postfix[i++]=item;
        }
        else if (item=='(')
        {
            push(item);
        }
        else if (item==')')
        {
            while (top != -1 && stack[top]!='(')
            {
                postfix[i++]=pop();
            }
            pop();
        }
        else
        {
            while (top!= -1 && prec(item)<=prec(stack[top]))
            {
                postfix[i++]=pop();
            }
        }
    }
}

```

```

        push(item);
    }
}
while(top!= -1)
{
    postfix[i++]=pop();
}
postfix[i] = '\0';
printf("\nPostfix Expression: \n %s", postfix);
}
int main()
{
    char infix[max];
    printf("\nEnter your infix expression: \n");
    scanf("%s",infix);
    infixtopostfix(infix);
    return 0;
}

```

Output:

C:\Users\STUDENT\Desktop\347\CircularQ\bin\Debug\CircularQ.exe

```

Enter your infix expression:
ab=(c*d)/e-f+g

Postfix Expression:
abcd*e/f-g+=
Process returned 0 (0x0)   execution time : 31.595 s
Press any key to continue.

```

Lab program 3A:

WAP 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.

```

#include <stdio.h>
#include <stdlib.h>
#define Max 5

void insert(int Queue[], int *front, int *rear, int item) {
    if (*rear == Max - 1) {
        printf("\nQueue Overflow!");
    }
}

```



```

    } else {
        if (*front == -1) {
            *front = 0;
        }
        Queue[++(*rear)] = item;
        printf("\nSuccessfully inserted %d", item);
    }
}

void delete(int Queue[], int *front, int *rear) {
    if (*front == -1 || *front > *rear) {
        printf("\nQueue Underflow");
    } else {
        printf("\nSuccessfully deleted %d", Queue[*front]);
        (*front)++;
        if (*front > *rear) {
            *front = *rear = -1;
        }
    }
}

void display(int Queue[], int *front, int *rear) {
    if (*front == -1 || *front > *rear) {
        printf("\nQueue is empty");
    } else {
        for (int i = *front; i <= *rear; i++) {
            printf("\nItem %d : %d", (i + 1), Queue[i]);
        }
    }
}

int main() {
    int Queue[Max];
    int front = -1;
    int rear = -1;
    int item, choice;

    while (1) {
        printf("\nEnter your choice:");
        printf("\n1 to Insert");
        printf("\n2 to Delete");
        printf("\n3 to Display");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("\nEnter item: ");
                scanf("%d", &item);
                insert(Queue, &front, &rear, item);
                break;
            case 2:

```

```
        delete(Queue, &front, &rear);
        break;
    case 3:
        display(Queue, &front, &rear);
        break;
    default:
        printf("\nInvalid choice !");
        break;
    }
}
}
```

Output:

C:\Users\STUDENT\Desktop\24\stack\win\Debug\stack.exe

```
Enter your choice:
1 to Insert
2 to Delete
3 to Display1

Enter item: 1

Successfully inserted 1
Enter your choice:
1 to Insert
2 to Delete
3 to Display2

Successfully deleted 1
Enter your choice:
1 to Insert
2 to Delete
3 to Display3

Queue is empty
Enter your choice:
1 to Insert
2 to Delete
3 to Display2

Queue Underflow
Enter your choice:
1 to Insert
2 to Delete
3 to Display1

Enter item: 1

Successfully inserted 1
Enter your choice:
1 to Insert
2 to Delete
3 to Display1

Enter item: 2

Successfully inserted 2
Enter your choice:
1 to Insert
2 to Delete
3 to Display1

Enter item: 3

Successfully inserted 3
Enter your choice:
1 to Insert
2 to Delete
3 to Display
1
```

```
Enter item: 4

Successfully inserted 4
Enter your choice:
1 to Insert
2 to Delete
3 to Display1

Enter item: 5

Successfully inserted 5
Enter your choice:
1 to Insert
2 to Delete
3 to Display1

Enter item: 6

Queue Overflow!
Enter your choice:
1 to Insert
2 to Delete
3 to Display_
```

LeetCode 1:

Implement Queue using Stacks

```
#include <stack>
using namespace std;

class MyQueue {
public:
    stack<int> s1, s2;

    MyQueue() {}

    void push(int x) {
        while (!s1.empty()) {
            s2.push(s1.top());
            s1.pop();
        }
        s1.push(x);
        while (!s2.empty()) {
            s1.push(s2.top());
            s2.pop();
        }
    }
};
```

```

    }
}

int pop() {
    if (s1.empty()) return -1;
    int ans = s1.top();
    s1.pop();
    return ans;
}

int peek() {
    return s1.empty() ? -1 : s1.top();
}

bool empty() {
    return s1.empty();
}
};

```

☒ Testcase
 ☒ Test Result

Accepted Runtime: 0 ms

• Case 1

Input

```
["MyQueue", "push", "push", "peek", "pop", "empty"]
```

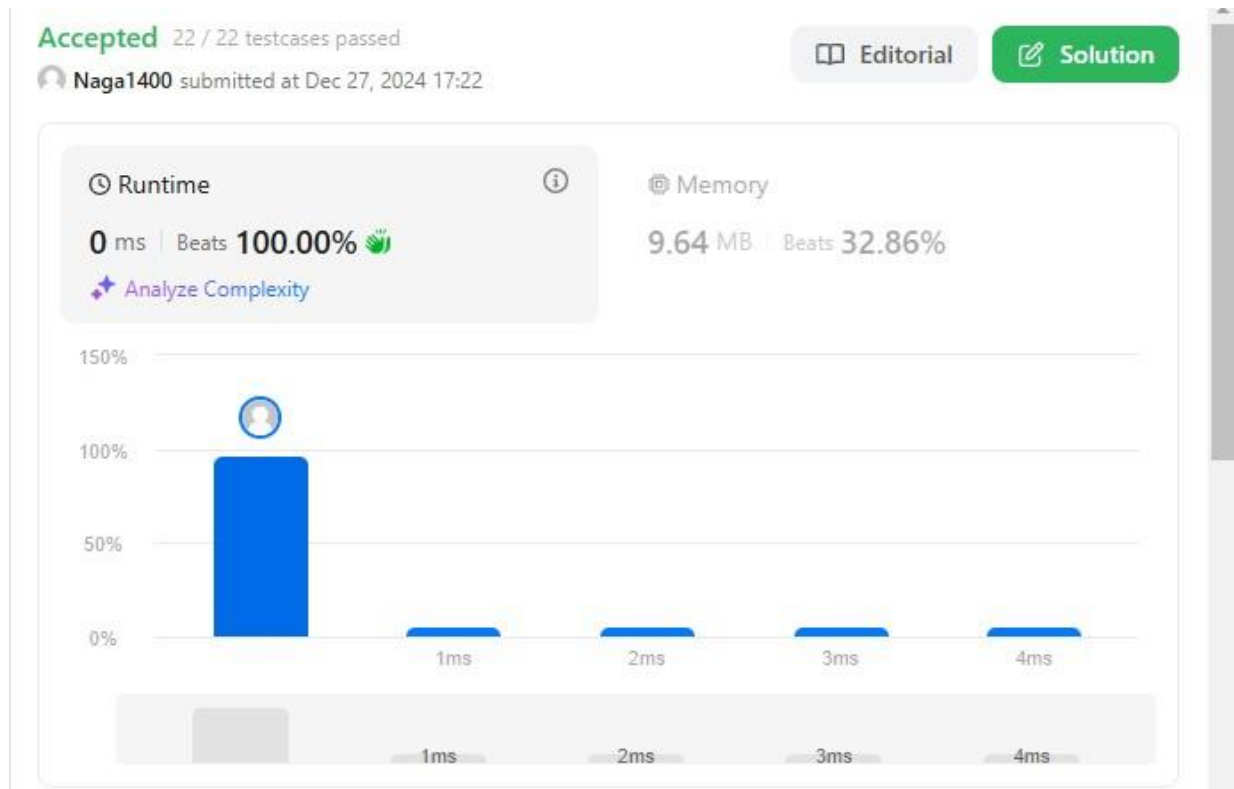
```
[[], [1], [2], [], [], []]
```

Output

```
[null, null, null, 1, 1, false]
```

Expected

```
[null, null, null, 1, 1, false]
```



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. The program should be done using pass by reference only.

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

#define MAX 5

typedef struct {
    int data[MAX];
    int front;
    int rear;
} CircularQueue;

void initializeQueue(CircularQueue *q) {
    q->front = -1;
    q->rear = -1;
}

int isFull(CircularQueue *q) {
    return (q->rear + 1) % MAX == q->front;
}
```

```

int isEmpty(CircularQueue *q) {
return q->front == -1;
}

void insert(CircularQueue *q, int value) {
if (isFull(q)) {
printf("Queue Overflow! Cannot insert %d\n", value);
return;
}
if (isEmpty(q)) {
q->front = 0;
}
q->rear = (q->rear + 1) % MAX;
q->data[q->rear] = value;
printf("Inserted %d into the queue\n", value);
}

int delete(CircularQueue *q) {
if (isEmpty(q)) {
printf("Queue Underflow! Cannot delete\n");
return -1;
}
int value = q->data[q->front];
if (q->front == q->rear) {
q->front = -1;
q->rear = -1;
} else {
q->front = (q->front + 1) % MAX;
}
printf("Deleted %d from the queue\n", value);
return value;
}

void display(CircularQueue *q) {
if (isEmpty(q)) {
printf("Queue is empty\n");
return;
}
printf("Queue elements: ");
int i = q->front;
while (1) {
printf("%d ", q->data[i]);
if (i == q->rear) break;
i = (i + 1) % MAX;
}
printf("\n");
}

int main() {
CircularQueue q;
initializeQueue(&q);

```

```
int choice, value;

while (1) {
printf("\n1 to Insert \n2 to Delete \n3 to Display \n4 to Exit");
printf("\nEnter your Choice : ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        printf("Enter value to insert: ");
        scanf("%d", &value);
        insert(&q, value);
        break;
    case 2:
        delete(&q);
        break;
    case 3:
        display(&q);
        break;
    case 4:
        printf("Exiting...\n");
        return 0;
    default:
        printf("Invalid choice, please try again.\n");
}
}
}
```

Output:

C:\Users\STUDENT\Desktop\347\CircularQ\bin\Debug\CircularQ.exe

```
1 to Insert
2 to Delete
3 to Display
4 to Exit
Choose an operation: 1
Enter value to insert: 10
Inserted 10 into the queue

1 to Insert
2 to Delete
3 to Display
4 to Exit
Choose an operation: 3
Queue elements: 10

1 to Insert
2 to Delete
3 to Display
4 to Exit
Choose an operation: 2
Deleted 10 from the queue

1 to Insert
2 to Delete
3 to Display
4 to Exit
Choose an operation: 2
Queue Underflow! Cannot delete

1 to Insert
2 to Delete
3 to Display
4 to Exit
Choose an operation:
```

LeetCode 2: Valid Parentheses

```
#include <stack>
#include <string>
using namespace std;

class Solution {
public:
    bool isValid(string s) {
        stack<char> st;
        for (int i = 0; i < s.length(); i++) {
            char ch = s[i];
```

```

        if (ch == '(' || ch == '{' || ch == '[') {
            st.push(ch);
        } else {
            if (!st.empty()) {
                char top = st.top();
                if ((ch == ')' && top == '(') ||
                    (ch == '}' && top == '{') ||
                    (ch == ']' && top == '[')) {
                    st.pop();
                } else {
                    return false;
                }
            } else {
                return false;
            }
        }
    }
    return st.empty();
}
};

```

☒ Testcase
 ☒ Test Result

Accepted
 Runtime: 2 ms

• Case 1
 • Case 2
 • Case 3
 • Case 4

Input

s =
 "()"

Output

true

Expected

true

Accepted 100 / 100 testcases passed

Naga1400 submitted at Dec 27, 2024 17:49

Editorial

Solution

Runtime

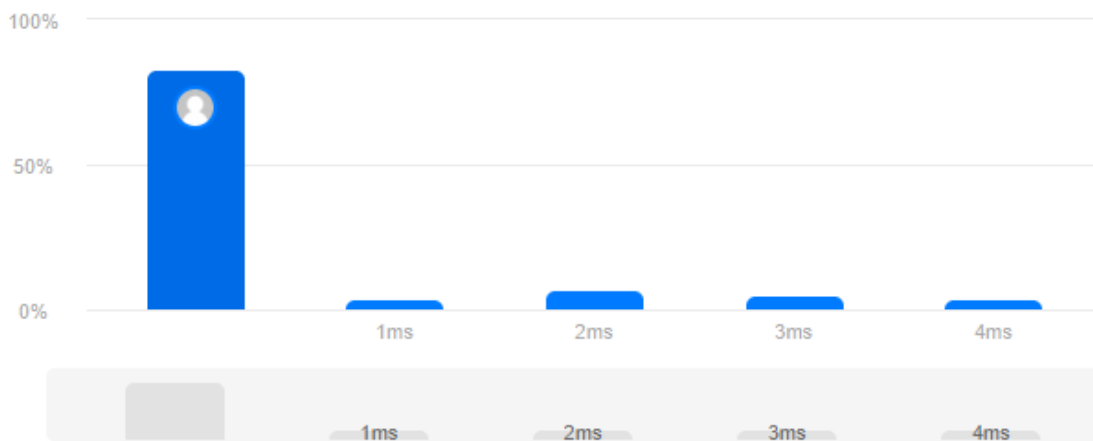


0 ms | Beats 100.00% 🌿

Analyze Complexity

Memory

8.61 MB | Beats 48.11%



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>
```

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

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

```

void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *head;
    *head = newNode;
}

```

```

void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* last = *head;
    while (last->next != NULL) {
        last = last->next;
    }
    last->next = newNode;
}

```

```

void insertAtPosition(struct Node** head, int data, int position) {
    struct Node* newNode = createNode(data);
    if (position == 0) {
        insertAtBeginning(head, data);
        return;
    }
    struct Node* current = *head;
    for (int i = 0; i < position - 1 && current != NULL; i++) {
        current = current->next;
    }
    if (current == NULL) {
        printf("Position out of bounds.\n");
        free(newNode);
        return;
    }
    newNode->next = current->next;
    current->next = newNode;
}

```

```

void deleteAtBeginning(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty.\n");
        return;
    }
    struct Node* temp = *head;
    *head = (*head)->next;
    free(temp);
}

```

```

void deleteByValue(struct Node** head, int value) {
    struct Node* current = *head;
    struct Node* prev = NULL;
    while (current != NULL && current->data != value) {
        prev = current;
        current = current->next;
    }
    if (current == NULL) {
        printf("Value %d not found in the list.\n", value);
        return;
    }
    if (prev == NULL) {
        *head = current->next;
    } else {
        prev->next = current->next;
    }
    free(current);
}

```

```

void deleteAtEnd(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty.\n");
        return;
    }
    struct Node* current = *head;
    struct Node* prev = NULL;
    while (current->next != NULL) {
        prev = current;
        current = current->next;
    }
    if (prev == NULL) {
        free(*head);
        *head = NULL;
    } else {
        prev->next = NULL;
        free(current);
    }
}

```

```

void displayList(struct Node* head) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    struct Node* current = head;
    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->next;
    }
}

```

```

    printf("NULL\n");
}

int main() {
    struct Node* head = NULL;
    int choice, data, position;


    while (1) {
        printf("\nSingly Linked List Operations:\n");
        printf("1. Insert at Beginning\n");
        printf("2. Insert at End\n");
        printf("3. Insert at Position\n");
        printf("4. Delete at Beginning\n");
        printf("5. Delete by Value\n");
        printf("6. Delete at End\n");
        printf("7. Display List\n");
        printf("8. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert at beginning: ");
                scanf("%d", &data);
                insertAtBeginning(&head, data);
                break;
            case 2:
                printf("Enter value to insert at end: ");
                scanf("%d", &data);
                insertAtEnd(&head, data);
                break;
            case 3:
                printf("Enter value to insert: ");
                scanf("%d", &data);
                printf("Enter position: ");
                scanf("%d", &position);
                insertAtPosition(&head, data, position);
                break;
            case 4:
                deleteAtBeginning(&head);
                break;
            case 5:
                printf("Enter value to delete: ");
                scanf("%d", &data);
                deleteByValue(&head, data);
                break;
            case 6:
                deleteAtEnd(&head);
                break;
            case 7:

```

```
        printf("Linked list contents:\n");
        displayList(head);
        break;
    case 8:
        exit(0);
    default:
        printf("Invalid choice. Please try again.\n");
    }
}
return 0;
}
```

Output:

 "C:\Users\STUDENT\Desktop\347\Linked list\bin\Debug\Linked list.exe"

Singly Linked List Operations:

1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete by Value
6. Delete at End
7. Display List
8. Exit

Enter your choice: 1

Enter value to insert at beginning: 10

Singly Linked List Operations:

1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete by Value
6. Delete at End
7. Display List
8. Exit

Enter your choice: 2

Enter value to insert at end: 100

Singly Linked List Operations:

1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete by Value
6. Delete at End
7. Display List
8. Exit

Enter your choice: 7

Linked list contents:

10 -> 100 -> NULL

Singly Linked List Operations:

1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete by Value
6. Delete at End
7. Display List
8. Exit

Enter your choice: 3

Enter value to insert: 50

Enter position: 1

Singly Linked List Operations:

1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete by Value
6. Delete at End
7. Display List
8. Exit

Enter your choice: 7

Linked list contents:

10 -> 50 -> 100 -> NULL

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>
```

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

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

```
void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *head;
    *head = newNode;
}
```

```
void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* last = *head;
    while (last->next != NULL) {
        last = last->next;
    }
    last->next = newNode;
}
```

```
void insertAtPosition(struct Node** head, int data, int position) {
    struct Node* newNode = createNode(data);
    if (position == 0) {
        insertAtBeginning(head, data);
    }
```

```

        return;
    }
    struct Node* current = *head;
    for (int i = 0; i < position - 1 && current != NULL; i++) {
        current = current->next;
    }
    if (current == NULL) {
        printf("Position out of bounds.\n");
        free(newNode);
        return;
    }
    newNode->next = current->next;
    current->next = newNode;
}

```

```

void deleteAtBeginning(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty.\n");
        return;
    }
    struct Node* temp = *head;
    *head = (*head)->next;
    free(temp);
}

```

```

void deleteByValue(struct Node** head, int value) {
    struct Node* current = *head;
    struct Node* prev = NULL;
    while (current != NULL && current->data != value) {
        prev = current;
        current = current->next;
    }
    if (current == NULL) {
        printf("Value %d not found in the list.\n", value);
        return;
    }
    if (prev == NULL) {
        *head = current->next;
    } else {
        prev->next = current->next;
    }
    free(current);
}

```

```

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

```

```

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

```

```

void displayList(struct Node* head) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    struct Node* current = head;
    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
}

```

```

int main() {
    struct Node* head = NULL;
    int choice, data, position;

    while (1) {
        printf("\nSingly Linked List Operations:\n");
        printf("1. Insert at Beginning\n");
        printf("2. Insert at End\n");
        printf("3. Insert at Position\n");
        printf("4. Delete at Beginning\n");
        printf("5. Delete by Value\n");
        printf("6. Delete at End\n");
        printf("7. Display List\n");
        printf("8. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert at beginning: ");
                scanf("%d", &data);


```

```

        insertAtBeginning(&head, data);
        break;
    case 2:
        printf("Enter value to insert at end: ");
        scanf("%d", &data);
        insertAtEnd(&head, data);
        break;
    case 3:
        printf("Enter value to insert: ");
        scanf("%d", &data);
        printf("Enter position: ");
        scanf("%d", &position);
        insertAtPosition(&head, data, position);
        break;
    case 4:
        deleteAtBeginning(&head);
        break;
    case 5:
        printf("Enter value to delete: ");
        scanf("%d", &data);
        deleteByValue(&head, data);
        break;
    case 6:
        deleteAtEnd(&head);
        break;
    case 7:
        printf("Linked list contents:\n");
        displayList(head);
        break;
    case 8:
        exit(0);
    default:
        printf("Invalid choice. Please try again.\n");
    }
}
return 0;
}

```

Output:

 "C:\Users\STUDENT\Desktop\347\Linked list\bin\Debug\Linked list.exe"

Singly Linked List Operations:

1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete by Value
6. Delete at End
7. Display List
8. Exit

Enter your choice: 1

Enter value to insert at beginning: 10

Singly Linked List Operations:

1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete by Value
6. Delete at End
7. Display List
8. Exit

Enter your choice: 2

Enter value to insert at end: 100

Singly Linked List Operations:

1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete by Value
6. Delete at End
7. Display List
8. Exit

Enter your choice: 7

Linked list contents:

10 -> 100 -> NULL

Singly Linked List Operations:

1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete by Value
6. Delete at End
7. Display List
8. Exit

Enter your choice: 3

Enter value to insert: 50

Enter position: 1


Singly Linked List Operations:

1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete by Value
6. Delete at End
7. Display List
8. Exit

Enter your choice: 7

Linked list contents:

10 -> 50 -> 100 -> NULL

 "C:\Users\STUDENT\Desktop\347\Linked list\bin\Debug\Linked list.exe"

Singly Linked List Operations:

1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete by Value
6. Delete at End
7. Display List
8. Exit

Enter your choice: 4

Singly Linked List Operations:

1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete by Value
6. Delete at End
7. Display List
8. Exit

Enter your choice: 7

Linked list contents:

50 -> 100 -> NULL

Singly Linked List Operations:

1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete by Value
6. Delete at End
7. Display List
8. Exit

Enter your choice: 6

Singly Linked List Operations:

1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete by Value
6. Delete at End
7. Display List
8. Exit

Enter your choice: 7

Linked list contents:

50 -> NULL

Singly Linked List Operations:

1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete by Value
6. Delete at End
7. Display List
8. Exit

Enter your choice: 5

Enter value to delete: 50

```
Singly Linked List Operations:
```

1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete by Value
6. Delete at End
7. Display List
8. Exit

```
Enter your choice: 5
```

```
Enter value to delete: 50
```

```
Singly Linked List Operations:
```

1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete by Value
6. Delete at End
7. Display List
8. Exit

```
Enter your choice: 7
```

```
Linked list contents:
```

```
List is empty.
```

```
Singly Linked List Operations:
```

1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete by Value
6. Delete at End
7. Display List
8. Exit

```
Enter your choice: _
```

Lab Program-6:

6a) WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists.

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

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

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

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

```

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

void append(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}

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

void sortList(struct Node* head) {
    if (head == NULL) return;
    struct Node* i = head;
    struct Node* j = NULL;
    int temp;
    while (i != NULL) {
        j = i->next;
        while (j != NULL) {
            if (i->data > j->data) {
                // Swap data
                temp = i->data;
                i->data = j->data;
                j->data = temp;
            }
            j = j->next;
        }
        i = i->next;
    }
}

void reverseList(struct Node** head) {
    struct Node *prev = NULL, *current = *head, *next = NULL;
    while (current != NULL) {
        next = current->next;

```



```

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

void concatenateLists(struct Node** head1, struct Node* head2) {
    if (*head1 == NULL) {
        *head1 = head2;
        return;
    }
    struct Node* temp = *head1;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = head2;
}

int main() {
    struct Node* list1 = NULL;
    struct Node* list2 = NULL;
    int n1, n2, data;

    printf("Enter the number of nodes for List 1: ");
    scanf("%d", &n1);
    printf("Enter the values for List 1:\n");
    for (int i = 0; i < n1; i++) {
        printf("Enter data for node %d: ", i + 1);
        scanf("%d", &data);
        append(&list1, data);
    }

    printf("Enter the number of nodes for List 2: ");
    scanf("%d", &n2);
    printf("Enter the values for List 2:\n");
    for (int i = 0; i < n2; i++) {
        printf("Enter data for node %d: ", i + 1);
        scanf("%d", &data);
        append(&list2, data);
    }

    int choice;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Sort List 1\n");
        printf("2. Reverse List 1\n");
        printf("3. Concatenate List 1 and List 2\n");
        printf("4. Print List 1\n");
        printf("5. Print List 2\n");
    }
}

```

```

printf("6. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        sortList(list1);
        printf("List 1 sorted.\n");
        break;
    case 2:
        reverseList(&list1);
        printf("List 1 reversed.\n");
        break;
    case 3:
        concatenateLists(&list1, list2);
        printf("List 2 concatenated to List 1.\n");
        break;
    case 4:
        printf("List 1: ");
        printList(list1);
        break;
    case 5:
        printf("List 2: ");
        printList(list2);
        break;
    case 6:
        printf("Exiting...\n");
        exit(0);
    default:
        printf("Invalid choice! Please try again.\n");
}
}

return 0;
}

```

Output:

"C:\Users\STUDENT\Desktop\D.s Lab code\SinglyLL\bin\Debug\SinglyLL.exe"

```
Enter the number of nodes for List 1: 3
Enter the values for List 1:
Enter data for node 1: 1
Enter data for node 2: 5
Enter data for node 3: 9
Enter the number of nodes for List 2: 3
Enter the values for List 2:
Enter data for node 1: 3
Enter data for node 2: 5
Enter data for node 3: 7
```

Menu:

1. Sort List 1
2. Reverse List 1
3. Concatenate List 1 and List 2
4. Print List 1
5. Print List 2
6. Exit

Enter your choice: 4

List 1: 1 -> 5 -> 9 -> NULL

Menu:

1. Sort List 1
2. Reverse List 1
3. Concatenate List 1 and List 2
4. Print List 1
5. Print List 2
6. Exit

Enter your choice: 2

List 1 reversed.

Menu:

1. Sort List 1
2. Reverse List 1
3. Concatenate List 1 and List 2
4. Print List 1
5. Print List 2
6. Exit

Enter your choice: 4

List 1: 9 -> 5 -> 1 -> NULL

Menu:

1. Sort List 1
2. Reverse List 1
3. Concatenate List 1 and List 2
4. Print List 1
5. Print List 2
6. Exit

Enter your choice: 3

List 2 concatenated to List 1.

```

Menu:
1. Sort List 1
2. Reverse List 1
3. Concatenate List 1 and List 2
4. Print List 1
5. Print List 2
6. Exit
Enter your choice: 4
List 1: 9 -> 5 -> 1 -> 3 -> 5 -> 7 -> NULL

Menu:
1. Sort List 1
2. Reverse List 1
3. Concatenate List 1 and List 2
4. Print List 1
5. Print List 2
6. Exit
Enter your choice: 1
List 1 sorted.

Menu:
1. Sort List 1
2. Reverse List 1
3. Concatenate List 1 and List 2
4. Print List 1
5. Print List 2
6. Exit
Enter your choice: 4
List 1: 1 -> 3 -> 5 -> 5 -> 7 -> 9 -> NULL

Menu:
1. Sort List 1
2. Reverse List 1
3. Concatenate List 1 and List 2
4. Print List 1
5. Print List 2
6. Exit
Enter your choice: 5
List 2: 5 -> 7 -> 9 -> NULL

```

6b) WAP to Implement Single Link List to simulate Stack & Queue Operations.

```

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

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

struct Node* createNode(int data) {

```

```

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

```

```

void push(struct Node** top, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *top;
    *top = newNode;
    printf("Pushed %d to stack\n", data);
}

```

```

void pop(struct Node** top) {
    if (*top == NULL) {
        printf("Stack is empty\n");
        return;
    }
    struct Node* temp = *top;
    *top = (*top)->next;
    printf("Popped %d from stack\n", temp->data);
    free(temp);
}

```

```

void enqueue(struct Node** front, struct Node** rear, int data) {
    struct Node* newNode = createNode(data);
    if (*rear == NULL) {
        *front = *rear = newNode;
        printf("Enqueued %d to queue\n", data);
        return;
    }
    (*rear)->next = newNode;
    *rear = newNode;
    printf("Enqueued %d to queue\n", data);
}

```

```

void dequeue(struct Node** front) {
    if (*front == NULL) {
        printf("Queue Underflow\n");
        return;
    }
    struct Node* temp = *front;
    int n=temp->data;
    *front = (*front)->next;
    free(temp);
    printf("Dequeued %d from queue\n", n);
}

```

```

void display(struct Node* head) {

```

```

    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
    while (temp != NULL) {
        printf("\n%d", temp->data);
        temp = temp->next;
    }
}

int main() {
    struct Node* stack = NULL;
    struct Node* front = NULL;
    struct Node* rear = NULL;
    int choice, data;

    while (1) {
        printf("\n1. Push to stack \n2. Pop from stack \n3. Display stack \n4. Enqueue to queue
\n5. Dequeue from queue \n6. Display queue \n7. Exit\nEnter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the data to push: ");
                scanf("%d", &data);
                push(&stack, data);
                break;
            case 2:
                pop(&stack);
                break;
            case 3:
                printf("Stack: ");
                display(stack);
                break;
            case 4:
                printf("Enter the data to enqueue: ");
                scanf("%d", &data);
                enqueue(&front, &rear, data);
                break;
            case 5:
                dequeue(&front);
                if (front == NULL) {
                    rear = NULL;
                }
                break;
            case 6:
                printf("Queue: ");
                display(front);
                break;
            case 7:

```

```

        exit(0);
    default:
        printf("Invalid choice\n");
    }
}
}

```

Output:

C:\Users\STUDENT\Desktop\DD158~1.SLA\SLAINS~1\bin\Debug\SLAINS~1.EXE

```

1. Push to stack
2. Pop from stack
3. Display stack
4. Enqueue to queue
5. Dequeue from queue
6. Display queue
7. Exit
Enter choice:
1
Enter the data to push: 10
Pushed 10 to stack

```

```

1. Push to stack
2. Pop from stack
3. Display stack
4. Enqueue to queue
5. Dequeue from queue
6. Display queue
7. Exit
Enter choice: 1
Enter the data to push: 20
Pushed 20 to stack

```

```

1. Push to stack
2. Pop from stack
3. Display stack
4. Enqueue to queue
5. Dequeue from queue
6. Display queue
7. Exit
Enter choice: 3
Stack:
20
10

```

```

1. Push to stack
2. Pop from stack
3. Display stack
4. Enqueue to queue
5. Dequeue from queue
6. Display queue
7. Exit
Enter choice: 2
Popped 20 from stack

```

```

1. Push to stack
2. Pop from stack
3. Display stack
4. Enqueue to queue
5. Dequeue from queue
6. Display queue
7. Exit
Enter choice: 3
Stack:
10

```

```
1. Push to stack
2. Pop from stack
3. Display stack
4. Enqueue to queue
5. Dequeue from queue
6. Display queue
7. Exit
Enter choice: 4
Enter the data to enqueue: 1
Enqueued 1 to queue
```

```
1. Push to stack
2. Pop from stack
3. Display stack
4. Enqueue to queue
5. Dequeue from queue
6. Display queue
7. Exit
Enter choice: 4
Enter the data to enqueue: 2
Enqueued 2 to queue
```

```
1. Push to stack
2. Pop from stack
3. Display stack
4. Enqueue to queue
5. Dequeue from queue
6. Display queue
7. Exit
Enter choice: 6
Queue:
```

```
1
```

```
2
```

```
1. Push to stack
2. Pop from stack
3. Display stack
4. Enqueue to queue
5. Dequeue from queue
6. Display queue
7. Exit
Enter choice: 5
Dequeued 1 from queue
```

```
1. Push to stack
2. Pop from stack
3. Display stack
4. Enqueue to queue
5. Dequeue from queue
6. Display queue
7. Exit
Enter choice: 6
Queue:
```

```
2
```


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>

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

struct node* head = NULL;

void create() {
    struct node *newnode, *temp;
    head = NULL;
    int choice = 1;

    while (choice) {
        newnode = (struct node*)malloc(sizeof(struct node));
        printf("Enter the data: ");
        scanf("%d", &newnode->data);

        newnode->next = NULL;
        newnode->prev = NULL;

        if (head == NULL) {
            head = temp = newnode;
        } else {
            temp->next = newnode;
            newnode->prev = temp;
            temp = newnode;
        }

        printf("Do you want to continue (type 0 or 1)? ");
        scanf("%d", &choice);
    }
}

void insertLeft(int value, int newValue) {
    struct node* temp = head;
```

41 | P a g e

```

struct node* newnode;

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

if (temp == NULL) {
    printf("Node with value %d not found.\n", value);
    return;
}

newnode = (struct node*)malloc(sizeof(struct node));
newnode->data = newvalue;
newnode->next = temp;
newnode->prev = temp->prev;

if (temp->prev != NULL) {
    temp->prev->next = newnode;
} else {
    head = newnode;
}
temp->prev = newnode;
}

void deleteNode(int value) {
    struct node* temp = head;

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

    if (temp == NULL) {
        printf("Node with value %d not found.\n", value);
        return;
    }

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

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

    free(temp);
    printf("Node with value %d deleted.\n", value);
}

void display() {

```

```

    struct node* temp = head;
    printf("Items in linked list are: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    int choice, value, new Value;

    do {
        printf("\n1. Create doubly linked list");
        printf("\n2. Insert a new node to the left of a node");
        printf("\n3. Delete a node based on value");
        printf("\n4. Display the list");
        printf("\n5. Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                create();
                break;
            case 2:
                printf("Enter the value of the node to insert left of: ");
                scanf("%d", &value);
                printf("Enter the new value to insert: ");
                scanf("%d", &new Value);
                insertLeft(value, new Value);
                break;
            case 3:
                printf("Enter the value of the node to delete: ");
                scanf("%d", &value);
                deleteNode(value);
                break;
            case 4:
                display();
                break;
            case 5:
                printf("Exiting program.\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 5);

    return 0;
}

```

Output:

```
1. Create doubly linked list
2. Insert a new node to the left of a node
3. Delete a node based on value
4. Display the list
5. Exit
Enter your choice: 1
Enter the data: 10
Do you want to continue (type 0 or 1)? 1
Enter the data: 20
Do you want to continue (type 0 or 1)? 1
Enter the data: 30
Do you want to continue (type 0 or 1)? 0

1. Create doubly linked list
2. Insert a new node to the left of a node
3. Delete a node based on value
4. Display the list
5. Exit
Enter your choice: 4
Items in linked list are: 10 20 30

1. Create doubly linked list
2. Insert a new node to the left of a node
3. Delete a node based on value
4. Display the list
5. Exit
Enter your choice: 2
Enter the value of the node to insert left of: 20
Enter the new value to insert: 15

1. Create doubly linked list
2. Insert a new node to the left of a node
3. Delete a node based on value
4. Display the list
5. Exit
Enter your choice: 4
Items in linked list are: 10 15 20 30

1. Create doubly linked list
2. Insert a new node to the left of a node
3. Delete a node based on value
4. Display the list
5. Exit
Enter your choice: 3
Enter the value of the node to delete: 10
Node with value 10 deleted.

1. Create doubly linked list
2. Insert a new node to the left of a node
3. Delete a node based on value
4. Display the list
5. Exit
Enter your choice: 4
Items in linked list are: 15 20 30

1. Create doubly linked list
2. Insert a new node to the left of a node
3. Delete a node based on value
4. Display the list
5. Exit
Enter your choice:
```


LeetCode : Middle of the Linked list


```
struct ListNode* middleNode(struct ListNode* head) {  
    struct ListNode* slow= head;  
    struct ListNode* fast= head;  
    while (fast != NULL && fast->next != NULL) {  
        slow = slow->next;  
        fast = fast->next->next;  
    }  
    return slow;  
}
```

Accepted 36 / 36 testcases passed

 Naga1400 submitted at Dec 27, 2024 18:43


 Editorial

 Solution

 Runtime

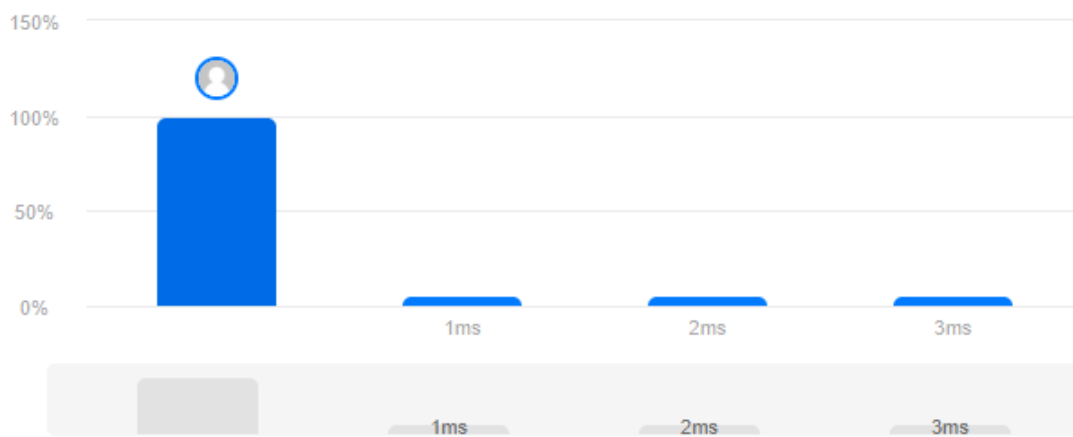


0 ms | Beats 100.00% 

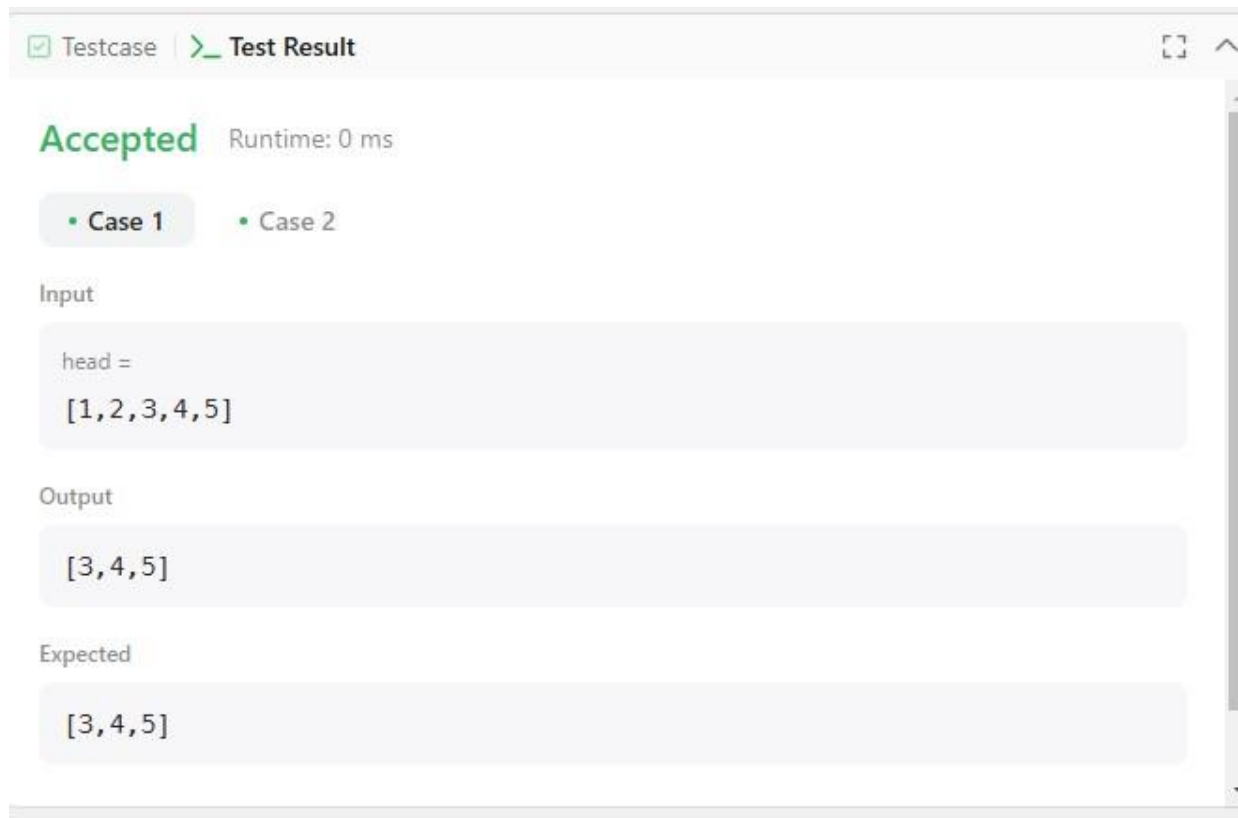
 Analyze Complexity

 Memory

8.34 MB | Beats 70.16% 



Code | C



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>
```

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

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

```

struct Node* insertNode(struct Node* root, int data) {
    if (root == NULL) {
        root = createNode(data);
    } else if (data < root->data) {
        root->left = insertNode(root->left, data);
    } else {
        root->right = insertNode(root->right, data);
    }
    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);
    }
}

```

```

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

    printf("Binary Search Tree Operations\n");
    printf("1. Insert an element\n");
    printf("2. In-order Traversal\n");
    printf("3. Pre-order Traversal\n");
    printf("4. Post-order Traversal\n");
    printf("5. Exit\n");

    while (1) {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);

```

```

switch (choice) {
    case 1:
        printf("Enter the element to insert: ");
        scanf("%d", &data);
        root = insertNode(root, data);
        break;
    case 2:
        if (root == NULL) {
            printf("Tree is empty.\n");
        } else {
            printf("In-order Traversal: ");
            inOrderTraversal(root);
            printf("\n");
        }
        break;
    case 3:
        if (root == NULL) {
            printf("Tree is empty.\n");
        } else {
            printf("Pre-order Traversal: ");
            preOrderTraversal(root);
            printf("\n");
        }
        break;
    case 4:
        if (root == NULL) {
            printf("Tree is empty.\n");
        } else {
            printf("Post-order Traversal: ");
            postOrderTraversal(root);
            printf("\n");
        }
        break;
    case 5:
        exit(0);
    default:
        printf("Invalid choice. Please try again.\n");
}

return 0;
}

```

Output:

C:\Users\STUDENT\Desktop\347\BinaryST\bin\Debug\BinaryST.exe

```
Binary Search Tree Operations
1. Insert an element
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit

Enter your choice: 1
Enter the element to insert: 20

Enter your choice: 1
Enter the element to insert: 5

Enter your choice: 1
Enter the element to insert: 30

Enter your choice: 1
Enter the element to insert: 8

Enter your choice: 1
Enter the element to insert: 9

Enter your choice: 1
Enter the element to insert: 50

Enter your choice: 2
In-order Traversal: 5 8 9 20 30 50

Enter your choice: 3
Pre-order Traversal: 20 5 8 9 30 50

Enter your choice: 4
Post-order Traversal: 9 8 5 50 30 20

Enter your choice: 5

Process returned 0 (0x0)   execution time : 380.106 s
Press any key to continue.
```

LeetCode :

Intersection of two linked list

```
struct ListNode *getIntersectionNode(struct ListNode *headA, struct ListNode *headB) {
    struct ListNode *a = headA, *b = headB;
    while (a != b) {
        a = (a == NULL) ? headB : a->next;
        b = (b == NULL) ? headA : b->next;
    }
    return a;
}
```

☒ Testcase | [Test Result](#)



Accepted Runtime: 0 ms

• **Case 1** • Case 2 • Case 3

Input

intersectVal =
8

listA =
[4,1,8,4,5]

listB =
[5,6,1,8,4,5]



skipA =
2

skipB =
3

Accepted 39 / 39 testcases passed

Naga1400 submitted at Dec 27, 2024 18:53

Editorial

Solution

Runtime



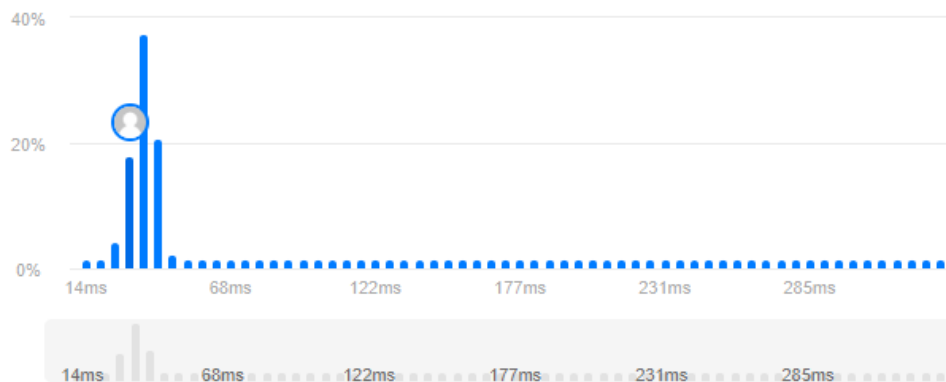
32 ms | Beats 89.98% 🌿

[Analyze Complexity](#)

Memory

17.52 MB | Beats 10.61%

[Analyze Complexity](#)



Code | C

Lab program-9:

9a) Write a program to traverse a graph using BFS method.

```
#include<stdio.h>
void bfs(int);
int a[10][10],vis[10],n;

void main()
{
    int i,j,src;

    printf("enter the number of vertices\n");
    scanf("%d",&n);
    printf("enter the adjacency matrix\n");
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            scanf("%d",&a[i][j]);

        }

        vis[i]=0;
    }

    printf("enter the src vertex\n");
    scanf("%d",&src);
    printf("nodes reachable from src vertex\n");
    bfs(src);
}

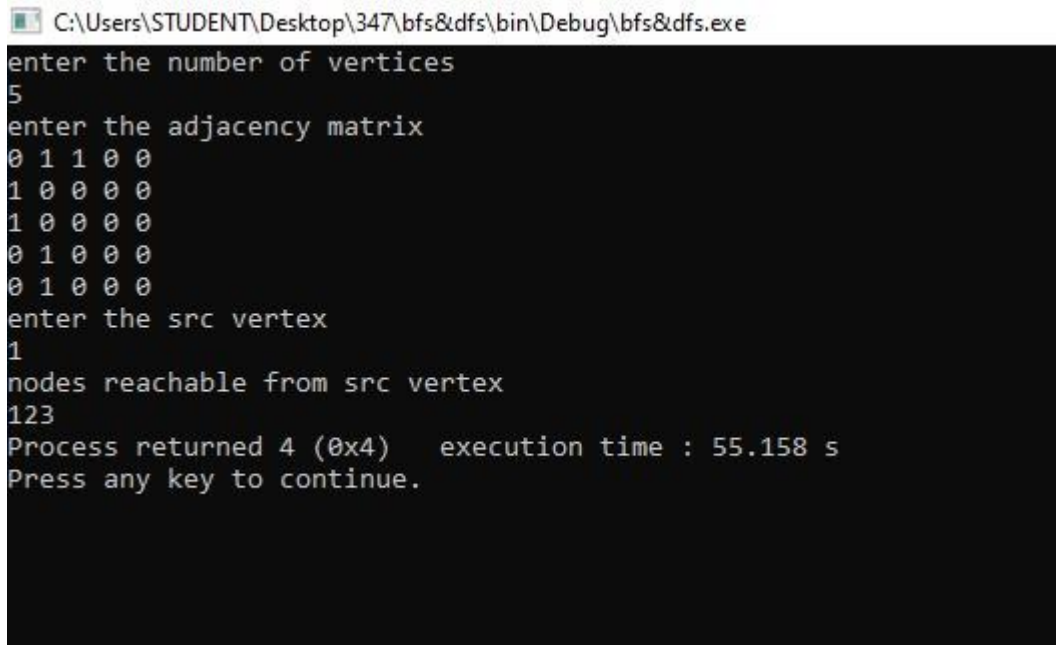
void bfs(int v)
{
    int q[10],f=1,r=1,u,i;
    q[r]=v;
    vis[v]=1;
    while(f<=r)
    {
        u=q[f];
        printf("%d",u);
        for(i=1;i<=n;i++)
        {
            if(a[v][i]==1 && vis[i]==0)
            {
                vis[i]=1;
                r=r+1;
                q[r]=i;
            }
        }
    }
}
```

```

    }
    f=f+1;
}
}

```

Output:



```

C:\Users\STUDENT\Desktop\347\bfs&dfs\bin\Debug\bfs&dfs.exe
enter the number of vertices
5
enter the adjacency matrix
0 1 1 0 0
1 0 0 0 0
1 0 0 0 0
0 1 0 0 0
0 1 0 0 0
enter the src vertex
1
nodes reachable from src vertex
123
Process returned 4 (0x4)   execution time : 55.158 s
Press any key to continue.

```

9b) Write a program to check whether given graph is connected or not using DFS method.

```

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

void dfs(int);
int n,i,j,a[10][10],vis[10];

void main()
{
    system("cls");
    printf("enter the number of vertices\n");
    scanf("%d",&n);
    printf("enter the adjacency matrix\n");
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            scanf("%d",&a[i][j]);
        }
        vis[i]=0;
    }
}

```

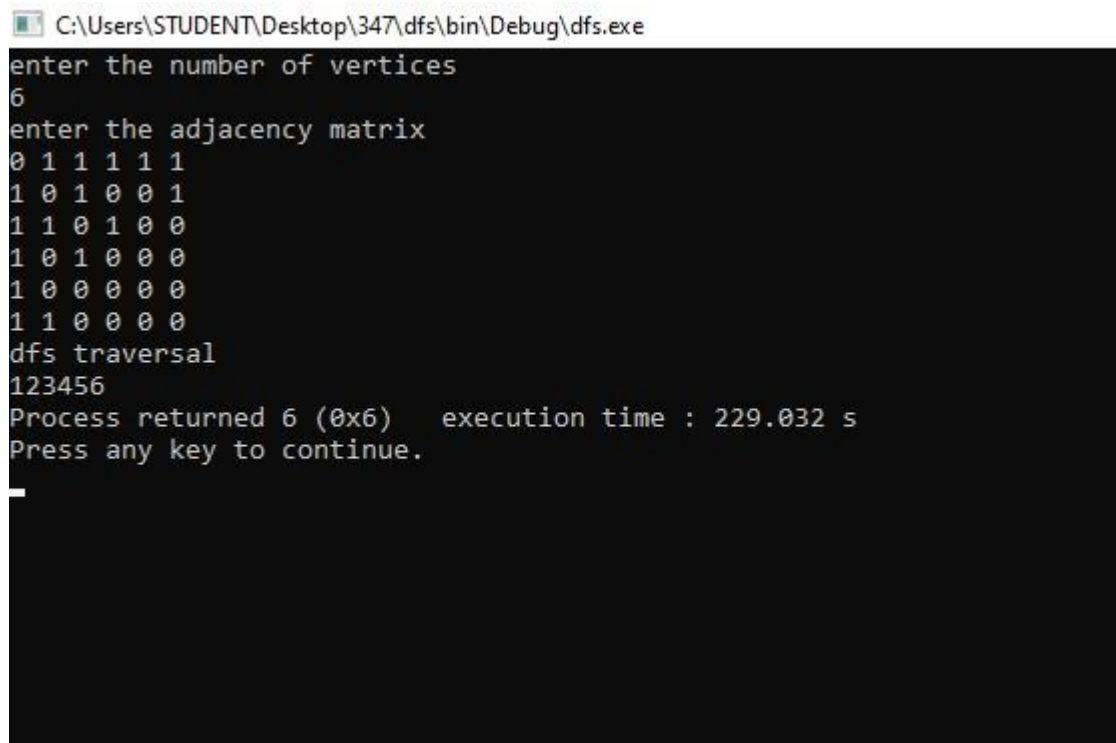
```

    }
    printf("dfs traversal\n");
    for(i=1;i<=n;i++)
    {
        if(vis[i]==0)
            dfs(i);
    }
}

void dfs(int v)
{
    vis[v]=1;
    printf("%d",v);
    for(j=1;j<=n;j++)
    {
        if(a[v][j]==1 && vis[j]==0)
        {
            dfs(j);
        }
    }
}
}

```

Output:



```

C:\Users\STUDENT\Desktop\347\dfs\bin\Debug\dfs.exe
enter the number of vertices
6
enter the adjacency matrix
0 1 1 1 1 1
1 0 1 0 0 1
1 1 0 1 0 0
1 0 1 0 0 0
1 0 0 0 0 0
1 1 0 0 0 0
dfs traversal
123456
Process returned 6 (0x6)   execution time : 229.032 s
Press any key to continue.

```

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>
int key[20],n,m;
int *ht,index;
int count = 0;
void insert(int key)
{
    index = key % m;
    while(ht[index] != -1)
    {
        index = (index+1)%m;
    }
    ht[index] = key;
    count++;
}
void display()
{
    int i;
    if(count == 0)
    {
        printf("&quot;\nHash Table is empty&quot;");
        return;
    }
    printf("&quot;\nHash Table contents are:\n &quot;");
    for(i=0; i<m; i++)
        printf("&quot;\n T[%d] --&gt; %d &quot;;", i, ht[i]);
}

void main()
{
    int i;
    printf("&quot;\nEnter the number of employee records (N) : &quot;");
    scanf("&quot;%d&quot;;", &n);

    printf("&quot;\nEnter the two digit memory locations (m) for hash table: &quot;");
    scanf("&quot;%d&quot;;", &m);
    ht = (int *)malloc(m*sizeof(int));
```

```

for(i=0; i<m; i++)
    ht[i] = -1;
printf(&quot;\nEnter the four digit key values (K) for N Employee Records:\n &quot;);
for(i=0; i<n; i++)
    scanf(&quot;%d&quot;, &amp;key[i]);
for(i=0; i<n; i++)
{
    if(count == m)
    {
        printf(&quot;\nHash table is full. Cannot insert the record %d key&quot;, i+1);
        break;
    }
    insert(key[i]);
}
display();
}

```

Output:

```

Enter the number of employee records (N): 5
Enter the two-digit memory locations (m) for hash table: 7
Enter the four-digit key values (K) for N Employee Records:
1234
5678
9101
1121
3141

```

Hash Table contents are:

T[0] --> 3141

T[1] --> 1121

T[2] --> 1234

T[3] --> -1

T[4] --> -1

T[5] --> 5678

T[6] --> 9101