

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT

On

DATA STRUCTURES (23CS3PCDST)

Submitted by

DAIVYA PRIYANKKUMAR SHAH (1BM23CS084)

**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 Daivya Priyankkumar Shah (**1BM23CS084**), 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.

Dr. Selva Kumar S
Associate Professor
Department of CSE
BMSCE, Bengaluru

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

Index Sheet

Sl. No.	Experiment Title	Page No.
1	Lab Program 1-Stacks	4-6
2	Lab Program 2-Application of Stacks	7-10
3	Lab Program 3-Queues	11-18
4	Lab Program 4-Singly Linked Lists	18-26
5	Lab Program 5-Linked Lists	26-31
6	Lab Program 6-Linked Lists and Applications	32-45
7	Lab Program 7-Doubly Linked Lists	45-49
8	Lab Program 8-Binary Search Tree	50-53
9	Lab Program 9-Graphs	53-57
10	Lab Program 10-Hashing	57-60

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 max 5

int stack[max];

int top = -1;

void push(int val){
    if(top==max-1){
        printf("overflow, stack is full\n");
    }
    else{
        top++;
        stack[top] = val;
    }
}

void pop(){
    if(top== -1){
        printf("underflow, stack is empty\n");
    }
    else{
        printf("element deleted:%d\n",stack[top]);
        top--;
    }
}

void display(){
    if(top== -1){
```

```
        printf("underflow, stack is empty\n");
    }
    else{
        for(int i=top;i>=0;i--){
            printf("%d ",stack[i]);
        }
        printf("\n");
    }
}

int main() {
    int choice, value;
    while(1) {
        printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch(choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push(value);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                return 0;
        }
    }
}
```

```
        default:
            printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
}
```

Output:

```
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 25
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 20
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 10
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
10 20 25
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
element deleted:10
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
20 25
```

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 1000
char stack[max];
int top=-1;
void push(char x){
    if(top==max-1){
        printf("stack full\n");
        return;
    }
    stack[++top]=x;
}
char pop(){
    if(top==-1){
        printf("stack empty\n");
        return -1;
    }
    return stack[top--];
}
int precedence(char x){
    if(x=='+'||x=='-'){
        return 1;
    }
    else if(x=='*'||x=='/'){
```

```

        return 2;
    }
    else if(x=='^'){
        return 3;
    }
    else{
        return 0;
    }
}

int isop(char x){
    return (x=='+'||x=='-'||x=='*'||x=='/'||x=='^');
}

void infixtopostfix(char *exp){
    char postfix[max];
    int i=0;
    char *ptr = exp;
    while(*ptr!='\0'){
        if(isalpha(*ptr)){
            postfix[i++] = *ptr;
        }
        if(*ptr=='('){
            push(*ptr);
        }
        if(*ptr==')'){
            while(stack[top]!='('){
                postfix[i++] = pop();
            }
            pop();
        }
    }
}

```



```

    if(isop(*ptr)){
        while(precedence(stack[top])>=precedence(*ptr)){
            postfix[i++]=pop();
        }
        push(*ptr);
    }
    ptr++;
}
while(top!=-1){
    postfix[i++]=pop();
}
postfix[i]='\0';
printf("Postfix:%s",postfix);
}

int main(){
    char exp[max];
    printf("enter expression:");
    scanf("%s",exp);
    infixtopostfix(exp);
    return 0;
}

```

Output:

```

enter expression:A+B+C+D
Postfix:AB+C+D+
PS C:\c++ practice\ac\stacks> ./infixtopostfix
enter expression:A+B*C+D
Postfix:ABC*+D+

```

Leetcode Program 1:

Moving Zeroes

```

void moveZeroes(int* nums, int numsSize) {
    int a=0;
    for(int i=0;i<numsSize;i++){
        if (nums[i]!=0){
            nums[a]=nums[i];
            a++;
        }
    }
    for(int i=a;i<numsSize;i++){
        nums[i]=0;
    }
}

```

← All Submissions

Accepted 74 / 74 testcases passed

daivya17 submitted at Dec 29, 2024 12:09

Editorial
Solution

⌚ Runtime
0 ms | Beats 100.00%
Analyze Complexity

💻 Memory
19.78 MB | Beats 19.14%

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1
Case 2

Input

```
nums =
[0,1,0,3,12]
```

Output

```
[1,3,12,0,0]
```

Expected

```
[1,3,12,0,0]
```

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
```

```
typedef struct {
```

```
    int items[MAX];
```

```
    int front;
```

```
    int rear;
```

```
} LinearQueue;
```

```
void initQueue(LinearQueue* q) {
```

```
    q->front = -1;
```

```
    q->rear = -1;
```

```
}
```

```
int isEmpty(LinearQueue* q) {
```

```
    return q->front == -1;
```

```
}
```

```
int isFull(LinearQueue* q) {
```

```
    return q->rear == MAX - 1;
```

```
}
```

```
void enqueue(LinearQueue* q, int value) {
```

```
    if (isFull(q)) {
```

```
        printf("Queue is full!\n");
```

```
        return;
```

```

    }
    if (isEmpty(q)) {
        q->front = 0;
    }
    q->rear++;
    q->items[q->rear] = value;
    printf("Enqueued: %d\n", value);
}

int dequeue(LinearQueue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty!\n");
        return -1;
    }
    int item = q->items[q->front];
    q->front++;
    if (q->front > q->rear) {
        q->front = q->rear = -1;
    }
    printf("Dequeued: %d\n", item);
    return item;
}

void display(LinearQueue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty!\n");
        return;
    }
    printf("Queue: ");
    for (int i = q->front; i <= q->rear; i++) {
        printf("%d ", q->items[i]);
    }
}

```

```
    }  
    printf("\n");  
}  
  
int main() {  
    LinearQueue q;  
    initQueue(&q);  
  
    int choice, value;  
  
    while (1) {  
        printf("1. Enqueue 2. Dequeue 3. Display 4. Exit\n");  
        printf("Enter your choice: ");  
        scanf("%d", &choice);  
  
        switch (choice) {  
            case 1:  
                printf("Enter value to enqueue: ");  
                scanf("%d", &value);  
                enqueue(&q, value);  
                break;  
            case 2:  
                dequeue(&q);  
                break;  
            case 3:  
                display(&q);  
                break;  
            case 4:  
                exit(0);  
        }  
    }  
}
```

```

        default:

            printf("Invalid choice! Please try again.\n");

        }

    }

    return 0;
}

```

Output:

```

1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice: 1
Enter value to enqueue: 1
Enqueued: 1
1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice: 1
Enter value to enqueue: 2
Enqueued: 2
1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice: 1
Enter value to enqueue: 3
Enqueued: 3
1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice: 3
Queue: 1 2 3
1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice: 2
Dequeued: 1
1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice: 2
Dequeued: 2
1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice: 2
Dequeued: 3
1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice: 2
Queue is empty!
1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice: 4

```

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

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

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

```
#define MAX 5
```

```
int circularQueue[MAX];
```

```
int front = -1, rear = -1;
```

```
void insert(int value) {
```

```
    if ((front == 0 && rear == MAX - 1) || (rear == (front - 1) % (MAX - 1))) {
```

```
        printf("Queue Overflow! Cannot insert %d\n", value);
```

```
        return;
```

```
    }
```

```
    if (front == -1) {
```

```
        front = rear = 0;
```

```
    } else if (rear == MAX - 1 && front != 0) {
```

```
        rear = 0;
```

```
    } else {
```

```
        rear++;
```

```
    }
```

```
    circularQueue[rear] = value;
```

```
    printf("Inserted %d into the queue\n", value);
```

```
}
```

```
void delete() {
```

```
    if (front == -1) {
```

```
        printf("Queue Underflow! No elements to delete\n");
```

```
        return;
    }
    printf("Deleted %d from the queue\n", circularQueue[front]);
    if (front == rear) {
        front = rear = -1;
    } else if (front == MAX - 1) {
        front = 0;
    } else {
        front++;
    }
}
```

```
void display() {
    if (front == -1) {
        printf("Queue is empty!\n");
        return;
    }
    printf("Queue elements: ");
    if (rear >= front) {
        for (int i = front; i <= rear; i++) {
            printf("%d ", circularQueue[i]);
        }
    } else {
        for (int i = front; i < MAX; i++) {
            printf("%d ", circularQueue[i]);
        }
        for (int i = 0; i <= rear; i++) {
            printf("%d ", circularQueue[i]);
        }
    }
}
```



```
    }  
}  
printf("\n");  
}  
  
int main() {  
    int choice, value;  
  
    while (1) {  
        printf("1. Enqueue 2. Dequeue 3. Display 4. Exit\n");  
        printf("Enter your choice: ");  
        scanf("%d", &choice);  
  
        switch (choice) {  
            case 1:  
                printf("Enter the value to insert: ");  
                scanf("%d", &value);  
                insert(value);  
                break;  
            case 2:  
                delete();  
                break;  
            case 3:  
                display();  
                break;  
            case 4:  
                exit(0);  
            default:  
                printf("Invalid choice! Please try again.\n");  
        }  
    }  
}
```

```

        break;

    }

}

return 0;

}

```

Output:

```

1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice: 1
Enter the value to insert: 10
Inserted 10 into the queue
1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice: 1
Enter the value to insert: 20
Inserted 20 into the queue
1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice: 1
Enter the value to insert: 30
Inserted 30 into the queue
1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice: 3
Queue elements: 10 20 30
1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice: 2
Deleted 10 from the queue
1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice: 3
Queue elements: 20 30
1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice: 4

```

Lab Program 4:

WAP to Implement Singly Linked List with following operations

a) Create a linked list.

b) Insertion of a node at first position, at any position and at end of list.

Display the contents of the linked list.

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

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

```
struct Node{
```

```
    int data;
```

```
    struct Node* next;
```

```

};

struct Node* createnode(int value){

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

    newNode->data=value;

    newNode->next=NULL;

    return newNode;

}

struct Node* createlinkedlist(){

    struct Node* head = NULL;

    return head;

}

void insertatstart(struct Node** head,int value){

    struct Node* newnode=createnode(value);

    newnode->next=*head;

    *head=newnode;

}

void insertatend(struct Node** head,int value){

    struct Node* newnode = createnode(value);

    if(*head==NULL){

        *head=newnode;

    }

    else{

        struct Node* temp = *head;

        while(temp->next!=NULL){

            temp=temp->next;

        }

        temp->next=newnode;

    }

}

```

```

void insertatposition(struct Node** head,int pos,int value){
    struct Node* newnode=createnode(value);
    if(pos==1){
        newnode->next=*head;
        *head=newnode;
    }
    else{
        struct Node* temp = *head;
        for (int i = 1; i < pos - 1 && temp != NULL; i++) {
            temp = temp->next;
        }
        if (temp == NULL) {
            printf("Position is greater than the length of the list.\n");
            return;
        }
        newnode->next = temp->next;
        temp->next = newnode;
    }
}

void display(struct Node* head){
    if(head==NULL){
        printf("linked list empty\n");
    }
    else{
        struct Node* temp=head;
        while(temp!=NULL){
            printf("%d ",temp->data);
            temp=temp->next;
        }
    }
}

```

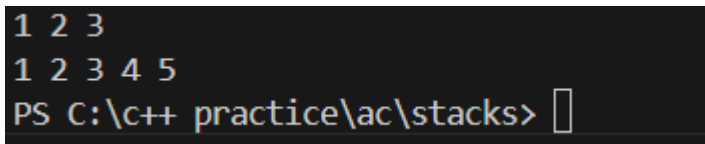
```

        printf("\n");
    }
}

int main(){
    struct Node* head = createlinkedlist();
    insertatstart(&head,3);
    insertatstart(&head,2);
    insertatstart(&head,1);
    display(head);
    insertatend(&head,5);
    insertatposition(&head,4,4);
    display(head);
    return 0;
}

```

Output:



```

1 2 3
1 2 3 4 5
PS C:\c++ practice\ac\stacks>

```

Leetcode Program 2:

Implement Queues using Stacks

```

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

typedef struct Stack {
    int *arr;
    int top;
    int capacity;
} Stack;

typedef struct {
    Stack *stack1;

```

```

    Stack *stack2;
} MyQueue;

Stack* createstack(int capacity){
    Stack *stack = (Stack *)malloc(sizeof(Stack));
    stack->capacity=capacity;
    stack->top=-1;
    stack->arr = (int *)malloc(capacity * sizeof(int));
    return stack;
}

bool isEmpty(Stack *stack) {
    return stack->top == -1;
}

void push(Stack *stack, int item){
    stack->arr[++stack->top] = item;
}

int pop(Stack *stack){
    return stack->arr[stack->top--];
}

int top(Stack *stack){
    return stack->arr[stack->top];
}

MyQueue* myQueueCreate() {
    MyQueue *queue = (MyQueue *)malloc(sizeof(MyQueue));
    queue->stack1=createstack(100);
    queue->stack2=createstack(100);
    return queue;
}

void myQueuePush(MyQueue* obj, int x) {

```

```

    push(obj->stack1,x);
}

int myQueuePop(MyQueue* obj) {
    if(isEmpty(obj->stack2)){
        while(!isEmpty(obj->stack1)){
            push(obj->stack2,pop(obj->stack1));
        }
    }
    return pop(obj->stack2);
}

int myQueuePeek(MyQueue* obj) {
    if(isEmpty(obj->stack2)){
        while(!isEmpty(obj->stack1)){
            push(obj->stack2,pop(obj->stack1));
        }
    }
    return top(obj->stack2);
}

bool myQueueEmpty(MyQueue* obj) {
    return isEmpty(obj->stack1) && isEmpty(obj->stack2);
}

void myQueueFree(MyQueue* obj) {
    free(obj->stack1->arr);
    free(obj->stack1);
    free(obj->stack2->arr);
}

```

```

    free(obj->stack2);

    free(obj);
}

/**
 * Your MyQueue struct will be instantiated and called as such:
 * MyQueue* obj = myQueueCreate();
 * myQueuePush(obj, x);

 * int param_2 = myQueuePop(obj);

 * int param_3 = myQueuePeek(obj);

 * bool param_4 = myQueueEmpty(obj);

 * myQueueFree(obj);
 */

```

Output:

Accepted 22 / 22 testcases passed

Editorial
Solution

daivya17 submitted at Oct 24, 2024 12:11

Runtime
0 ms | Beats 100.00%
Analyze Complexity

Memory
8.15 MB | Beats 78.78%

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]

Leetcode Program 3:

Backspace String Compare

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

```
#include <string.h>
```

```
bool backspaceCompare(char* s, char* t) {
```

```
    char s1[1000];
```

```
    char s2[1000];
```

```
    int i = 0, j = 0;
```

```
    for (int k = 0; k < strlen(s); k++) {
```

```
        if (s[k] == '#') {
```

```
            if (i > 0) {
```

```
                --i;
```

```
            }
```

```
        } else {
```

```
            s1[i++] = s[k];
```

```
        }
```

```
    }
```

```
    s1[i] = '\0';
```

```
    for (int k = 0; k < strlen(t); k++) {
```

```
        if (t[k] == '#') {
```

```
            if (j > 0) {
```

```
                --j;
```

```
            }
```

```
        } else {
```

```
            s2[j++] = t[k];
```

```
        }
```

```

    }

    s2[j] = '\0';

    return strcmp(s1, s2) == 0;
}

```

Output:

The screenshot shows a submission page for a problem. At the top, it says "Accepted" with "114 / 114 testcases passed". The user "daivya17" submitted it on Nov 07, 2024 at 12:50. There are buttons for "Editorial" and "Solution". Below this, a summary box shows "Runtime: 0 ms | Beats 100.00%" and "Memory: 8.25 MB | Beats 10.05%". A link to "Analyze Complexity" is also present. The "Test Result" section shows "Accepted" with "Runtime: 0 ms". It lists three cases: "Case 1", "Case 2", and "Case 3". Under "Input", it shows "s = 'ab#c'" and "t = 'ad#c'". Under "Output", it shows "true".

Lab Program 5:

WAP to Implement Singly Linked List with following operations

- Create a linked list.
- 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 val){

    struct Node* newnode=(struct Node*) malloc (sizeof(struct Node));

    newnode->data=val;

    newnode->next=NULL;

    return newnode;

}

struct Node* createll(){

    struct Node* head=NULL;

    return head;

}

void insertatstart(struct Node** head,int data){

    struct Node* node1=createnode(data);

    node1->next=*head;

    *head=node1;

}

void insertatend(struct Node** head,int data){

    struct Node* node1=createnode(data);

    if(*head==NULL){

        *head=node1;

    }

}
```

```

else{

    struct Node* temp=*head;

    while(temp->next!=NULL){

        temp=temp->next;

    }

    temp->next=node1;

}

}

void display(struct Node** head){

    struct Node* temp=*head;

    while(temp!=NULL){

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

        temp=temp->next;

    }

    printf("\n");

}

void deletefirst(struct Node** head){

    struct Node* temp=*head;

    *head=temp->next;

    temp->next=NULL;

    free(temp);

}

void deletelast(struct Node** head){

    struct Node* temp = *head;

    struct Node* prev = NULL;

    if (temp->next == NULL) {

        *head = NULL;

```

```
        free(temp);

        return;
    }

    while (temp->next != NULL) {

        prev = temp;

        temp = temp->next;
    }

    prev->next = NULL;

    free(temp);
}

void deletebyval(struct Node** head,int val){

    struct Node* temp = *head;

    if(temp!=NULL && temp->data==val){

        *head=temp->next;

        free(temp);
    }

    struct Node* prev=NULL;

    while(temp!=NULL && temp->data!=val){

        prev=temp;

        temp=temp->next;
    }

    if(temp==NULL){

        printf("element not found\n");
    }

    prev->next=temp->next;

    free(temp);
}
```

```

int main(){

    struct Node* head=createll();

    insertatstart(&head,5);

    insertatstart(&head,4);

    insertatstart(&head,3);

    insertatstart(&head,2);

    insertatstart(&head,1);

    display(&head);

    deletefirst(&head);

    display(&head);

    deletelast(&head);

    display(&head);

    deletebyval(&head,3);

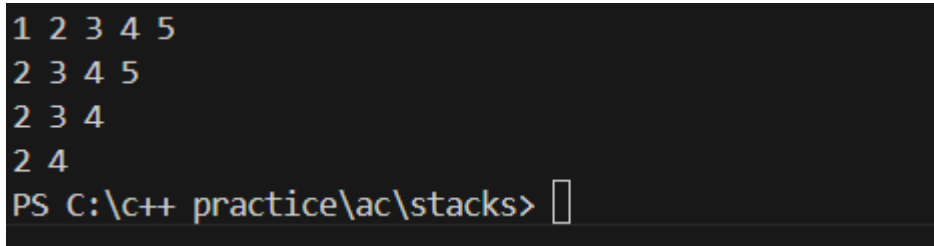
    display(&head);

    return 0;

}

```

Output:



```

1 2 3 4 5
2 3 4 5
2 3 4
2 4
PS C:\c++ practice\ac\stacks>

```

Leetcode Program 4:

Delete Duplicates

```

/**

* Definition for singly-linked list.

* struct ListNode {

*     int val;

```

```

*   struct ListNode *next;
* };
*/

struct ListNode* deleteDuplicates(struct ListNode* head) {
    struct ListNode* temp=head;
    while(temp!=NULL && temp->next!=NULL){
        if(temp->val==temp->next->val){
            temp->next=temp->next->next;
        }
        else{
            temp=temp->next;
        }
    }
    return head;
}

```

Output:

Accepted 168 / 168 testcases passed

daivya17 submitted at Nov 14, 2024 11:54

Editorial
Solution

Runtime
0 ms | Beats 100.00%

Memory
10.92 MB | Beats 55.59%

Analyze Complexity

Testcase
Test Result

Accepted Runtime: 0 ms

Case 1

Case 2

Input

head =
[1,1,2]

Output

[1,2]

Expected

[1,2]

Lab Program 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 val){
```

```
    struct Node* newnode=(struct Node*) malloc (sizeof(struct Node));
```

```
    newnode->data = val;
```

```
    newnode->next = NULL;
```

```
    return newnode;
```

```
}
```

```
struct Node* createll(){
```

```
    struct Node* head = NULL;
```

```
    return head;
```

```
}
```

```
void insertatstart(struct Node** head,int val){
```

```
    struct Node* newnode = createnode(val);
```

```
    newnode->next = *head;
```

```
    *head = newnode;
```

```
}
```

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

```
    struct Node* temp = head;
```

```
    while(temp!=NULL){
```



```
        printf("%d ",temp->data);

        temp = temp->next;

    }

    printf("\n");

}

int len(struct Node* head){

    int count = 0;

    struct Node* temp = head;

    while(temp!=NULL){

        temp = temp->next;

        count++;

    }

    return count;

}

void sort(struct Node** head){

    int n = len(*head);

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

        struct Node* temp = *head;

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

            if(temp->data > temp->next->data){

                int x = temp->data;

                temp->data = temp->next->data;

                temp->next->data = x;

            }

            temp = temp->next;

        }

    }

}
```

```

}

void rev(struct Node** head){
    struct Node* prev = NULL;
    struct Node* curr = *head;
    struct Node* forward = NULL;
    while(curr!=NULL){
        forward = curr->next;
        curr->next = prev;
        prev = curr;
        curr = forward;
    }
    *head = prev;
}

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

int main(){
    struct Node* head1 = createll();
    insertatstart(&head1,3);
    insertatstart(&head1,2);
    insertatstart(&head1,4);
    insertatstart(&head1,5);
    insertatstart(&head1,1);

```

```
printf("List1:\n");
display(head1);
printf("After reversing:\n");
rev(&head1);
display(head1);
printf("After sorting:\n");
sort(&head1);
display(head1);
struct Node* head2 = createll();
insertatstart(&head2,10);
insertatstart(&head2,9);
insertatstart(&head2,8);
insertatstart(&head2,7);
insertatstart(&head2,6);
printf("List2:\n");
display(head2);
printf("After concatenation:\n");
concat(&head1,&head2);
display(head1);
return 0;
}
```

Output:

```
List1:
1 5 4 2 3
After reversing:
3 2 4 5 1
After sorting:
1 2 3 4 5
List2:
6 7 8 9 10
After concatenation:
1 2 3 4 5 6 7 8 9 10
PS C:\c++ practice\ac\stacks> █
```

Lab Program 6b:

WAP to Implement Single Link List to simulate Stack & Queue

Operations.

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

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

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

```
s_node* init(int data) {
    s_node *n = (s_node*)malloc(sizeof(s_node));
    if (n == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }
    n->data = data;
    n->next = NULL;
```

```

        return n;
    }

s_node* push(s_node **last, int data, s_node **head) {
    if (*head == NULL) {

        *head = init(data);

        *last = *head;

        return *head;
    }

    s_node *l = *last;

    l->next = init(data);

    *last = l->next;

    return l->next;
}

s_node* pop(s_node **head) {
    if (*head == NULL) {

        printf("List is empty, nothing to pop.\n");

        return NULL;
    }

    s_node *h = *head;

    if (h->next == NULL) {

        free(h);

        *head = NULL;
    }
}

```

```
    return NULL;
}
```

```
while (h->next->next != NULL) {
    h = h->next;
}
```

```
s_node *temp = h->next;
free(temp);
h->next = NULL;
return h;
}
```

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

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

```
}
```

```
int main() {
```

```
    s_node *head = NULL;
```

```
    s_node *last = NULL;
```

```
    int choice, data;
```

```
    while (1) {
```

```
        printf("\nMenu:\n");
```

```
        printf("1. Push a node\n");
```

```
        printf("2. Pop a node\n");
```

```
        printf("3. Print the list\n");
```

```
        printf("4. Exit\n");
```

```
        printf("Enter your choice: ");
```

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

```
        switch (choice) {
```

```
            case 1:
```

```
                printf("Enter the data to push: ");
```

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

```
                last = push(&last, data, &head);
```

```
                break;
```

```
            case 2:
```

```
                last = pop(&head);
```

```
                break;
```

case 3:

```
print_list(head);
```

```
break;
```

case 4:

```
while (head != NULL) {
```

```
    head = pop(&head);
```

```
}
```

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

```
return 0;
```

default:

```
printf("Invalid choice! Please try again.\n");
```

```
}
```

```
}
```

```
}
```

Output:


```
1. Push a node
2. Pop a node
3. Print the list
4. Exit
Enter your choice: 1
Enter the data to push: 10

Menu:
1. Push a node
2. Pop a node
3. Print the list
4. Exit
Enter your choice: 1
Enter the data to push: 20

Menu:
1. Push a node
2. Pop a node
3. Print the list
4. Exit
Enter your choice: 1
Enter the data to push: 30

Menu:
1. Push a node
2. Pop a node
3. Print the list
4. Exit
Enter your choice: 2

Menu:
1. Push a node
2. Pop a node
3. Print the list
4. Exit
Enter your choice: 3
10 -> 20 -> NULL
```

Queues using linked lists

```
#include <stdio.h>

#include <stdlib.h>

typedef struct p_node {
    int data;

    struct p_node* next;
} p_node;

p_node* init(int data) {
    p_node *n = (p_node*)malloc(sizeof(p_node));

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

```

    }

    n->data = data;

    n->next = NULL;

    return n;
}

p_node* enqueue(p_node **last, int data, p_node **head) {

    if (*head == NULL) {

        *head = init(data);

        *last = *head;

        return *head;

    }

    p_node *l = *last;

    l->next = init(data);

    *last = l->next;

    return l->next;

}

int dequeue(p_node **head) {

    if (*head == NULL) {

        printf("Queue is empty, nothing to dequeue.\n");

        return -1;

    }

    p_node *h = *head;

    int r = h->data;

    *head = h->next;

    free(h);

    return r;

}

```

```
void print_list(p_node *head) {
    if (head == NULL) {
        printf("Queue is empty.\n");
        return;
    }
    p_node *current = head;
    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
}

int main() {
    p_node *head = NULL;
    p_node *last = NULL;
    int choice, data;
    while (1) {
        printf("\nMenu:\n");
        printf("1. Enqueue a node\n");
        printf("2. Dequeue a node\n");
        printf("3. Print the queue\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter the data to enqueue: ");
```

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

last = enqueue(&last, data, &head);

break;

case 2:

    data = dequeue(&head);

    if (data != -1) {

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

    }

    break;

case 3:

    print_list(head);

    break;

case 4:

    while (head != NULL) {

        dequeue(&head);

    }

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

    return 0;

default:

    printf("Invalid choice! Please try again.\n");

}

}

}
```

Output:

```
Menu:
1. Enqueue a node
2. Dequeue a node
3. Print the queue
4. Exit
Enter your choice: 1
Enter the data to enqueue: 10
```

```
Menu:
1. Enqueue a node
2. Dequeue a node
3. Print the queue
4. Exit
Enter your choice: 1
Enter the data to enqueue: 20
```

```
Menu:
1. Enqueue a node
2. Dequeue a node
3. Print the queue
4. Exit
Enter your choice: 1
Enter the data to enqueue: 30
```

```
Menu:
1. Enqueue a node
2. Dequeue a node
3. Print the queue
4. Exit
Enter your choice: 2
Dequeued: 10
```

```
Menu:
1. Enqueue a node
2. Dequeue a node
3. Print the queue
4. Exit
Enter your choice: 3
20 -> 30 -> NULL
```

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* prev;

    struct Node* next;
};

struct Node* createNode(int val) {
    struct Node* n = (struct Node*)malloc(sizeof(struct Node));

    n->data = val;

    n->prev = n->next = NULL;

    return n;
}

void insertEnd(struct Node** head, int val) {
    struct Node* n = createNode(val);

    if (!*head) {
        *head = n;

        return;
    }

    struct Node* t = *head;

    while (t->next) t = t->next;

    t->next = n;

    n->prev = t;
}

void insertLeft(struct Node** head, int target, int val) {
    struct Node* t = *head;

    while (t && t->data != target) t = t->next;

```

```

    if (!t) return;

    struct Node* n = createNode(val);

    n->next = t;

    n->prev = t->prev;

    if (t->prev) t->prev->next = n;

    else *head = n;

    t->prev = n;
}

void deleteNode(struct Node** head, int val) {

    struct Node* t = *head;

    while (t && t->data != val) t = t->next;

    if (!t) return;

    if (t->prev) t->prev->next = t->next;

    else *head = t->next;

    if (t->next) t->next->prev = t->prev;

    free(t);

}

void display(struct Node* head) {

    while (head) {

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

        head = head->next;

    }

    printf("NULL\n");

}

```

```

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

    insertEnd(&dll, 10);
    insertEnd(&dll, 20);
    insertEnd(&dll, 30);
    printf("Doubly Linked List: ");
    display(dll);

    insertLeft(&dll, 20, 15);
    printf("After Inserting 15 to the left of 20: ");
    display(dll);

    deleteNode(&dll, 10);
    printf("After Deleting 10: ");
    display(dll);

    return 0;
}

```

Output:

```

Doubly Linked List: 10 <-> 20 <-> 30 <-> NULL
After Inserting 15 to the left of 20: 10 <-> 15 <-> 20 <-> 30 <-> NULL
After Deleting 10: 15 <-> 20 <-> 30 <-> NULL

```

Leetcode Program 5:

Linked List Cycle

```
/**
```

```
* Definition for singly-linked list.
```

```
* struct ListNode {
```



```

*   int val;

*   struct ListNode *next;

* };

*/

bool hasCycle(struct ListNode *head) {

    struct ListNode* slow=head;

    struct ListNode* fast=head;

    while(fast && fast->next){

        slow=slow->next;

        fast=fast->next->next;

        if(slow==fast){

            return true;

        }

    }

    return false;

}

```

Output:

Accepted
29 / 29 testcases passed

daivya17 submitted at Nov 14, 2024 12:32

Editorial
Solution

Runtime
13 ms | Beats 29.55%

Memory
11.26 MB | Beats 34.75%

Testcase
Test Result

Accepted
Runtime: 3 ms

Case 1
Case 2
Case 3

Input

head =
[3,2,0,-4]

pos =
1

Output

true

Lab Program 8:

Write a program

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

```
#include<stdio.h>

#include<stdlib.h>

typedef struct BST{
    int data;
    struct BST* left;
    struct BST* right;
}node;

node* create(){
    node* newnode = (node*)malloc(sizeof(node));
    printf("enter data:");
    scanf("%d",&newnode->data);
    newnode->left = newnode->right = NULL;
    return newnode;
}

void insert(node* root,node* newnode){
    if(newnode->data < root->data){
        if(root->left!=NULL){
            insert(root->left,newnode);
        }
        else{
            root->left = newnode;
        }
    }
```

```

    }

    if(newnode->data > root->data){
        if(root->right!=NULL){
            insert(root->right,newnode);
        }
        else{
            root->right = newnode;
        }
    }
}

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

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

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

```

```
        preorder(root->left);
        preorder(root->right);
    }
}

int main() {
    char ch;

    node *root = NULL, *temp;

    do {
        temp = create();
        if (root == NULL)
            root = temp;
        else
            insert(root, temp);

        printf("\nDo you want to enter more(y/n)? ");
        getchar(); // To consume the newline character from previous input
        scanf("%c", &ch);
    } while (ch == 'y' || ch == 'Y');

    printf("\nPreorder Traversal: ");
    preorder(root);

    printf("\nInorder Traversal: ");
    inorder(root);

    printf("\nPostorder Traversal: ");
```

```
    postorder(root);

    return 0;
}
```

Output:

```
enter data:20

Do you want to enter more(y/n)? y
enter data:10

Do you want to enter more(y/n)? y
enter data:5

Do you want to enter more(y/n)? y
enter data:25

Do you want to enter more(y/n)? y
enter data:30

Do you want to enter more(y/n)? n

Preorder Traversal: 20 10 5 25 30
Inorder Traversal: 5 10 20 25 30
Postorder Traversal: 5 10 30 25 20
```

Lab Program 9a:

Write a program to traverse a graph using BFS method.

```
#include<stdio.h>

#include<conio.h>

int adj[10][10];

int n;

int vis[10];

void bfs(int v){

    int q[10],rear=1,front=1,u;
```

```
vis[v] = 1;

q[rear] = v;

while(front<=rear){

    u = q[front];

    printf("%d ",u);

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

        if(adj[u][i]==1 && vis[i]==0){

            vis[i] = 1;

            rear++;

            q[rear] = i;

        }

    }

    front++;

}

int main(){

    printf("enter no of vertices:");

    scanf("%d",&n);

    printf("enter adjacency matrix:");

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

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

            scanf("%d",&adj[i][j]);

        }

        vis[i] = 0;

    }

    int v;

    printf("enter node to start traversing:");
```

```

scanf("%d",&v);

bfs(v);

}

```

Output:

```

enter no of vertices:4
enter adjacency matrix:0 1 1 0 1 0 0 1 1 0 0 1 0 1 1 0
enter node to start traversing:2
2 1 4 3
PS C:\c++ practice\ac\stacks>

```

Lab Program 9b:

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

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

```
#define MAX 10
```

```
int adj[MAX][MAX], visited[MAX], n;
```

```
void dfs(int v) {
```

```
    visited[v] = 1;
```

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

```
        if (adj[v][i] != 0 && visited[i] == 0) {
```

```
            dfs(i);
```

```
        }
```

```
    }
```

```
}
```

```
int main() {  
  
    printf("Enter the number of vertices: ");  
  
    scanf("%d", &n);  
  
  
    printf("Enter the adjacency matrix (0 for no edge):\n");  
  
    for (int i = 1; i <= n; i++) {  
  
        for (int j = 1; j <= n; j++) {  
  
            scanf("%d", &adj[i][j]);  
  
        }  
    }  
  
  
    // Initialize the visited array  
  
    for (int i = 1; i <= n; i++) {  
  
        visited[i] = 0;  
  
    }  
  
  
    // Start DFS from the first vertex  
  
    dfs(1);  
  
  
    // Check if all vertices are visited  
  
    int connected = 1;  
  
    for (int i = 1; i <= n; i++) {  
  
        if (visited[i] == 0) {  
  
            connected = 0;  
  
            break;  
  
        }  
    }  
}
```



```

if (connected) {

    printf("The graph is connected.\n");

} else {

    printf("The graph is disconnected.\n");

}

return 0;

}

```

Output:

```

Enter the number of vertices: 4
Enter the adjacency matrix (0 for no edge):
0 1 1 0 1 0 0 1 1 0 0 1 1 0
The graph is connected.
PS C:\c++ practice\ac\stacks>

```

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() {
```

```
    if (count == 0) {
```

```
        printf("\nHash Table is empty");
```

```
        return;
```

```
    }
```

```
    printf("\nHash Table contents are:\n");
```

```
    for (int i = 0; i < m; i++) {
```

```
        printf("\nT[%d] --> %d", i, ht[i]);
```

```
    }
```

```
}
```

```
void main() {
```

```
    printf("\nEnter the number of employee records (N): ");
```

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

```
printf("\nEnter the two-digit memory locations (m) for hash table: ");
```

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

```
ht = (int *)malloc(m * sizeof(int));
```

```
for (int i = 0; i < m; i++)
```

```
    ht[i] = -1;
```

```
printf("\nEnter the four-digit key values (K) for %d Employee Records:\n", n);
```

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

```
    scanf("%d", &key[i]);
```

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

```
    if (count == m) {
```

```
        printf("\nHash table is full. Cannot insert record %d key", i + 1);
```

```
        break;
```

```
    }
```

```
    insert(key[i]);
```

```
}
```

```
display();
```

```
free(ht);
```

```
}
```

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 5 Employee Records:

1234

2345

4567

5678

6789

Hash Table contents are:

T[0] --> 2345

T[1] --> 5678

T[2] --> 1234

T[3] --> 4567

T[4] --> -1

T[5] --> -1

T[6] --> 6789

PS C:\c++ practice\ac\stacks> █