VISVESVARAYA TECHNOLOGICAL UNIVERSITY

**“Jnana Sangama”, Belgaum -590014, Karnataka.**



# LAB REPORT

**On**

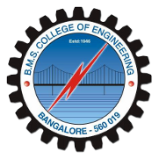
# DATA STRUCTURES (23CS3PCDST)

**Submitted by Dhanush K (24BECS405)**

**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 Dhanush k **(24BECS405)**, 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 Dr. Kavitha Sooda**

Associate Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

**Index Sheet**

|  |  |  |
| --- | --- | --- |
| **Sl.**  **No.** | **Experiment Title** | **Page No.** |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |
| 6 |  |  |
| 7 |  |  |
| 8 |  |  |
| 9 |  |  |
| 10 |  |  |

**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:**

1. **Push**
2. **Pop**
3. **Display**

**The program should print appropriate messages for stack overflow, stack underflow.**

#include <stdio.h>

#define size 2

void push(int);

void pop();

void display();

int stack[size],top = -1;

void main(){

int value, choice;

while(1){

printf("\n\n\*\*\*\*\*Choice\*\*\*\*\*\n");

printf("1. push\n");

printf("2. pop\n");

printf("3. Display\n");

printf("Enter the your choice: ");

scanf("%d",&choice);

switch(choice){

case 1:

printf("Enter the value to be insert: ");

scanf("%d",&value);

push(value);

break;

case 2: pop();

break;

case 3: display();

break;

default:

printf("wrong selection!!!!");

break;

}

}

}

void push (int value){

if(top == size-1)

printf("\nStack is Fulll!!! Insertion is not possible!!1");

else{

top++;

stack[top] = value;

printf("\nInsertion success!!!");

}

}

void pop(){

if(top == -1)

printf("\nStack is empty!!! Deletion is not possible!!");

else{

printf("\nDeleted\n: %d", stack[top]);

top--;

}

}

void display(){

if (top == -1)

printf("\nStack is emptyyy!!");

else{

int i;

printf("\nStack elements are:\n");

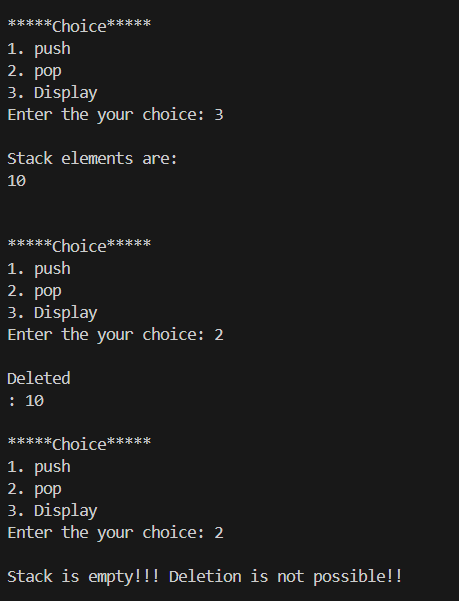
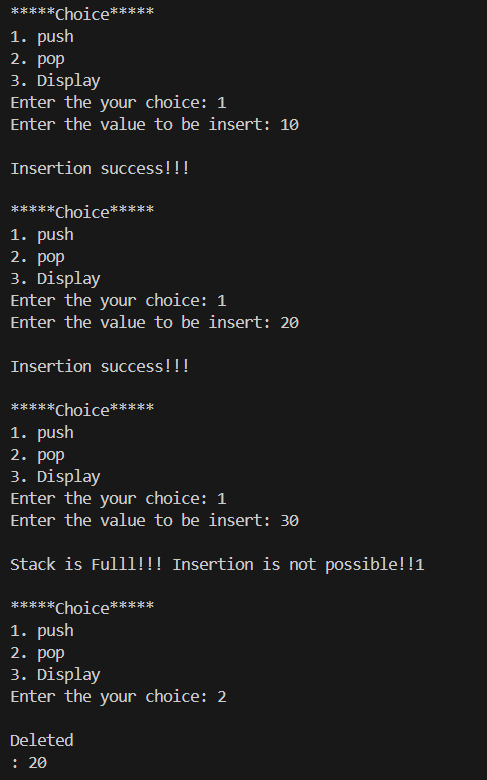
for(i=top; i>=0; i--)

printf("%d\n", stack[i]);

}

}

**Output:**

****

**Lab program 2:**

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

#include <stdio.h>

#include <ctype.h>

#define SIZE 50

char stack [SIZE];

int top=-1;

void push (char elem){

stack[++top]=elem;

}

char pop(){

return (stack[top--]);

}

int pr(char symbol){

if (symbol == '^'){

return 3;

}

else if (symbol == '\*'||symbol == '/'){

return 2;

}

else if (symbol == '+'||symbol == '-'){

return 1;

}

else{

return 0;

}

}

void main (){

char infix[50] ,postfix[50],ch,elem;

int i=0,k=0;

printf("Enter the infix exp: ");

scanf("%s",infix);

push('#');

while ((ch=infix[i++]) != '\0'){

if (ch == '('){

push(ch);

}

else if (isalnum(ch)){

postfix[k++]=ch;

}

else if (ch == ')'){

while ((stack[top]) != '('){

postfix[k++] = pop();

}

elem = pop();

}

else{

while (pr(stack[top]) >= pr(ch)){

postfix[k++] = pop();

}

push(ch);

}

}

while (stack[top] != '#'){

postfix[k++] = pop();

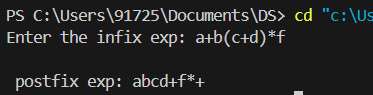
}

postfix[k] = '\0';

printf("\n postfix exp: %s\n",postfix);

}

**Output:**

****

**Lab program 3:**

**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 SIZE 3

int queue[SIZE];

int front = -1, rear = -1;

void insert(int);

void delete();

void display();

void main() {

int value, choice;

while (1) {

printf("\n\*\*\*\* Menu \*\*\*\*\n");

printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the number to insert into the queue: ");

scanf("%d", &value);

insert(value);

break;

case 2:

delete();

break;

case 3:

display();

break;

case 4:

exit(0);

break;

default:

printf("Wrong Selection!!!\n");

break;

}

}

}

void insert(int value) {

if (rear == SIZE - 1) {

printf("Overflow!!!\n");

} else

{

if (front == -1) {

front = 0;

}

rear++;

queue[rear] = value;

printf("Insertion Success!!!\n");

}

}

void delete() {

if (front == -1) {

printf("Underflow!!!\n");

} else {

printf("Deleted: %d\n", queue[front]);

if (front == rear) {

front = rear = -1;

front++;

}

}

}

void display() {

if (front == -1) {

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

} else {

printf("Elements in the queue: ");

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

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

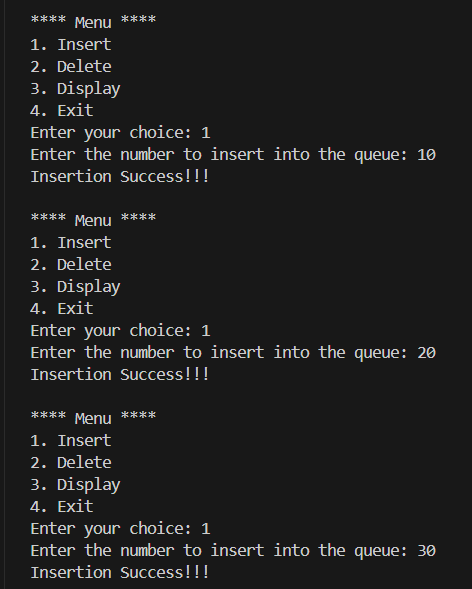
}

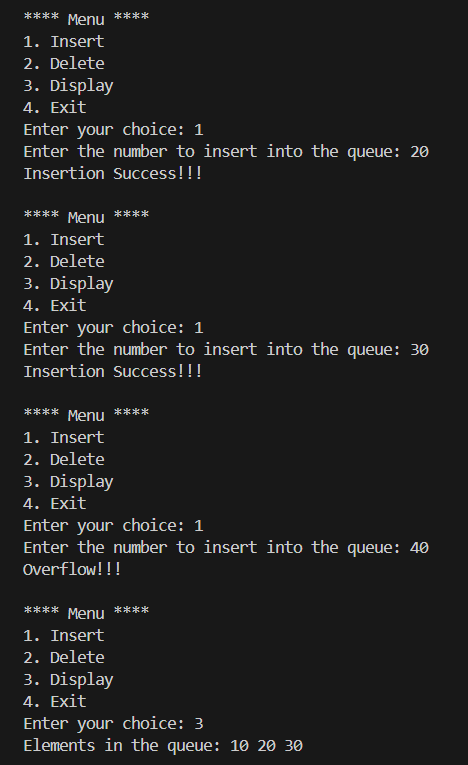
printf("\n");

}

}

**Output:**

****

****

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

#include <stdio.h>

#include <stdlib.h>

#define SIZE 3

int queue[SIZE];

int front = -1, rear = -1;

void insert(int);

void delete();

void display();

void main() {

int value, choice;

while (1) {

printf("\n\*\*\*\* Menu \*\*\*\*\n");

printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the number to insert into the queue: ");

scanf("%d", &value);

insert(value);

break;

case 2:

delete();

break;

case 3:

display();

break;

case 4:

exit(0);

break;

default:

printf("Wrong Selection!!!\n");

break;

}

}

}

void insert(int value) {

if ((rear+1)%SIZE == front) {

printf("Overflow!!!\n");

} else {

if (front == -1) {

front = 0;

}

rear=(rear+1)%SIZE;

queue[rear] = value;

printf("Insertion Success!!!\n");

}

}

void delete() {

if (front == -1) {

printf("Underflow!!!\n");

} else {

printf("Deleted: %d\n", queue[front]);

if (front == rear) {

front = rear = -1;

}

else{

front = (front+1) %SIZE;

}

}

}

void display() {

if (front == -1) {

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

} else {

printf("Elements in the queue: ");

int i = front;

while (i != rear) {

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

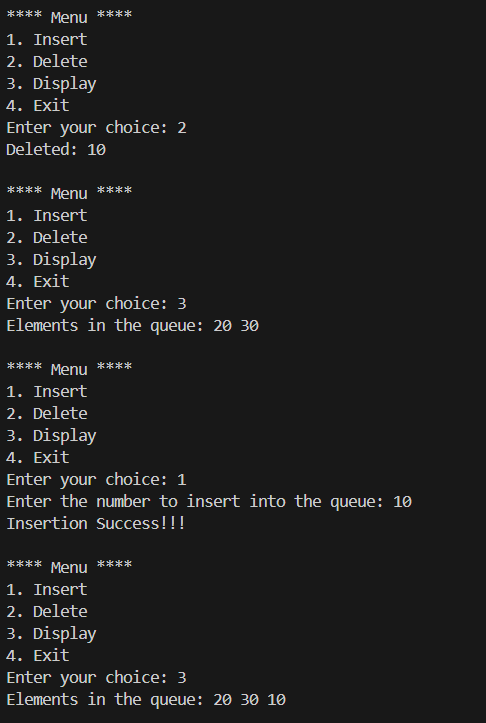
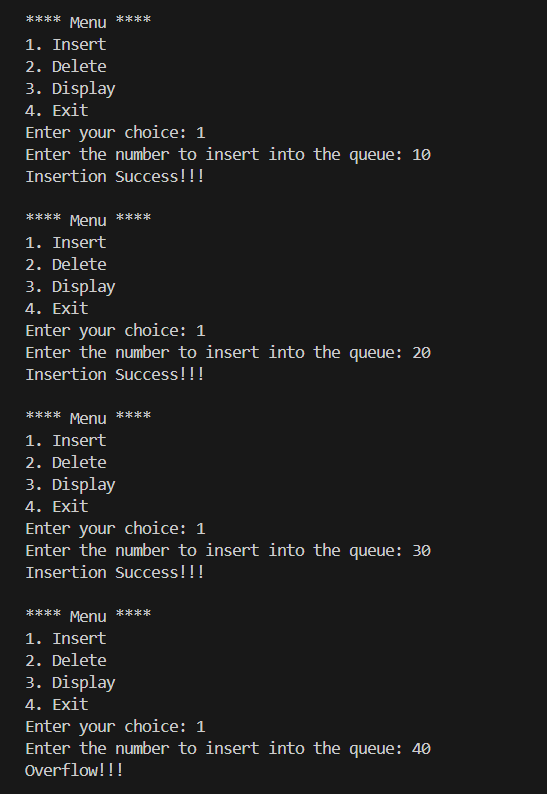
i = (i + 1) % SIZE;

}

printf("%d\n", queue[rear]);

}

**Output:**

****

**Lab program 4:**

**Leet code (**[**First Unique Character in a String**](https://leetcode.com/problems/first-unique-character-in-a-string/)**)**

int firstUniqChar(char \* s){

    int i = 0, map[26], sSize = strlen(s);

    for(i; i < 26; i++)

        map[i] = 0;

    for(i=0; i < sSize; i++)

        map[s[i] - 'a']++;

    for(i=0; i < sSize; i++)

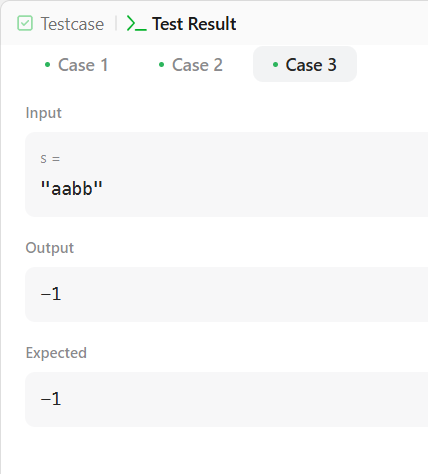
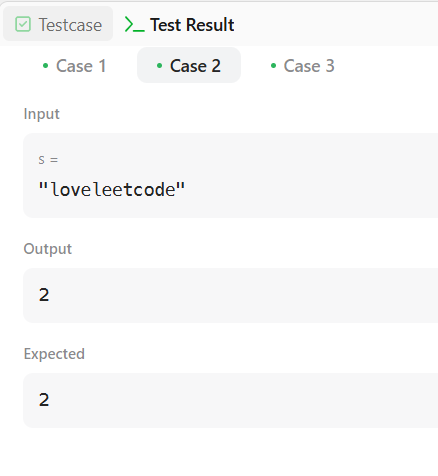
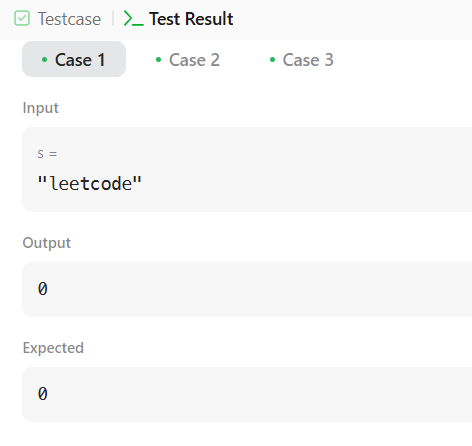
        if (map[s[i] - 'a'] == 1)

            return i;

    return -1;

}

**Output:**

****

**Lab program 5:**

**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 data) {

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

newNode-> data = data;

newNode->next = NULL;

return newNode;

}

struct Node\* createList() {

return NULL;

}

void insertAtFirst(struct Node\*\* head, int data) {

struct Node\* newNode = createNode(data);

newNode->next = \*head;

\*head = newNode;

printf("Node %d inserted at the beginning.\n", data);

}

void insertAtEnd(struct Node\*\* head, int data) {

struct Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;

printf("Node %d inserted at the end.\n", data);

return;

}

struct Node\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

printf("Node %d inserted at the end.\n", data);

}

void insertAtPosition(struct Node\*\* head, int data, int position) {

if (position < 1) {

printf("Invalid position.\n");

return;

}

struct Node\* newNode = createNode(data);

if (position == 1) {

newNode->next = \*head;

\*head = newNode;

printf("Node %d inserted at position %d.\n", data, position);

return;

}

struct Node\* temp = \*head;

for (int i = 1; i < position - 1 && temp != NULL; i++) {

temp = temp->next;

}

if (temp == NULL) {

printf("Position out of bounds.\n");

return;

}

newNode->next = temp->next;

temp->next = newNode;

printf("Node %d inserted at position %d.\n", data, position);

}

void displayList(struct Node\* head) {

if (head == NULL) {

printf("The list is empty.\n");

return;

}

struct Node\* temp = head;

printf("List: ");

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

}

int main() {

struct Node\* head = createList();

int choice, data, position;

while (1) {

printf("\nMenu:\n");

printf("1. Create a linked list\n");

printf("2. Insert at the first position\n");

printf("3. Insert at any position\n");

printf("4. Insert at the end\n");

printf("5. Display the linked list\n");

printf("6. Exit\n");

printf(“Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

head = createList();

printf("Linked list created.\n");

break;

case 2:

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

scanf("%d", &data);

insertAtFirst(&head, data);

break;

case 3:

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

scanf("%d", &data);

printf("Enter the position: ");

scanf("%d", &position);

insertAtPosition(&head, data, position);

break;

case 4:

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

scanf("%d", &data);

insertAtEnd(&head, data);

break;

case 5:

displayList(head);

break;

case 6:

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

exit(0);

break;

default:

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

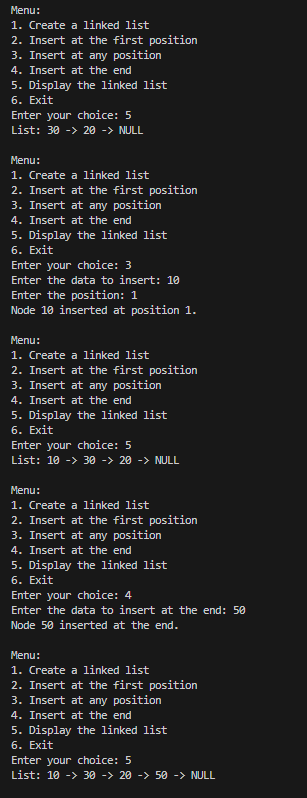
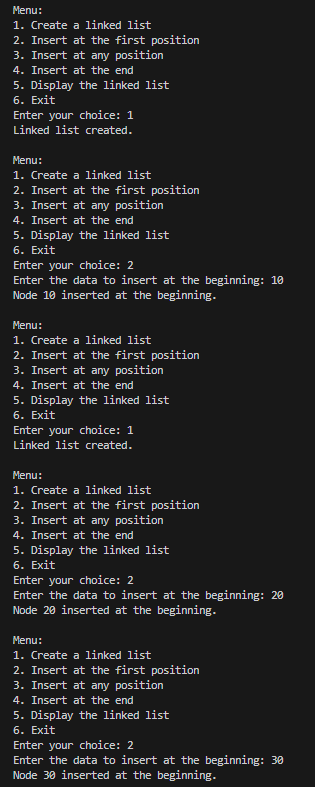
}

}

return 0;

}

**Output:**

****

**Leet code-1 (**[**Backspace String Compare**](https://leetcode.com/problems/backspace-string-compare/)**)**

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

    int nS=0, nT=0;

    register int i, j;

    for(i=strlen(s)-1, j=strlen(t)-1; i>=0||j>=0; i--, j--){

        while(i>=0){

            if (s[i]=='#') nS++, i--;

            else if (nS>0) nS--, i--;

            else break;

        }

        while(j>=0){

            if (t[j]=='#') nT++, j--;

            else if (nT>0) nT--, j--;

            else break;

        }

        if (i>=0 && j>=0 && s[i]!=t[j]) return 0;

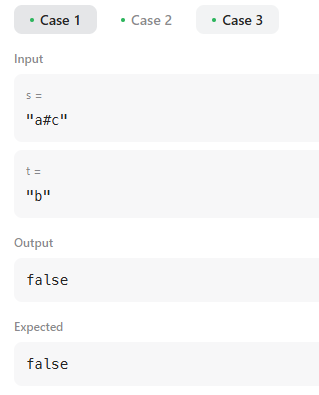
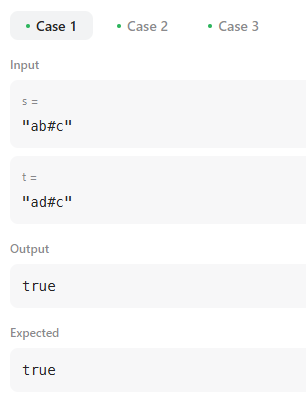
        if ((i>=0)^(j>=0)) return 0;

    }

    return 1;

}

**Output:**

****

**Leet code-2 (**[**Remove Digit From Number to Maximize**](https://leetcode.com/problems/remove-digit-from-number-to-maximize-result/)**)**

char\* removeDigit(char\* number, char digit) {

int len = strlen(number);

char \*s1 = malloc(len + 1);

char \*max = malloc(len + 1);

int k = 0, l = 0, list[len];

for(int i = 0; i < len; i++)

if(number[i] == digit)

list[l++] = i;

for(int i = 0; i < l; i++)

{

k = 0;

for(int j = 0; j < len; j++)

{

if(j == list[i])

continue;

s1[k++] = number[j];

}

s1[k] = '\0';

for(int m = 0; m < strlen(s1); m++)

{

if(s1[m] > max[m]) strcpy(max, s1);

else if(max[m] > s1[m]) break;

}

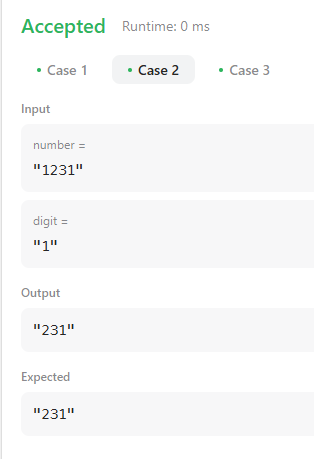
}

free(s1);

return max;

}

**Output:**

****

**Lab program 6:**

**WAP to Implement Singly Linked List with following operations**

**a) Create a linked list.**

**b) Deletion of first element, specified element andlast 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 \* new\_node ;

new\_node = (struct node \*) malloc(sizeof(struct node));

new\_node->data = data;

new\_node->next = NULL;

return new\_node;

}

struct node\* insertion(struct node \* head, int data){

struct node \* new\_node = createnode(data);

new\_node->next = head;

head = new\_node;

return head;

}

struct node\* DB(struct node \* head){

struct node\* ptr;

if (head==NULL){

printf("Empty");

}

else{

ptr=head;

head=head->next;

free(ptr);

}

return head;

}

struct node\* DE(struct node \* head){

struct node\* ptr;

struct node\* prev;

if (head==NULL){

printf("Empty");

}

else if (head->next == NULL){

free(head);

head=NULL;

}

else{

ptr = head;

while(ptr->next != NULL){

prev = ptr;

ptr= ptr->next;

}

prev->next=NULL;

free(ptr);

}

return head;

}

struct node\* DP(struct node \*head, int pos) {

if (head == NULL) {

printf("Empty list\n");

return NULL;

}

if (pos == 1) {

struct node\* temp = head;

head = head->next;

free(temp);

return head;

}

struct node\* ptr = head;

struct node\* prev = NULL;

for (int i = 1; ptr != NULL && i < pos; i++) {

prev = ptr;

ptr = ptr->next;

}

if (ptr == NULL) {

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

return head;

}

prev->next = ptr->next;

free(ptr);

return head;

}

struct node\* display(struct node \* head){

struct node \* temp = head;

while(temp != NULL){

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

temp= temp->next;

}

printf("Null\n");

}

void main(){

struct node\* head=NULL;

head= insertion(head,10);

head= insertion(head,20);

head= insertion(head,30);

head= insertion(head,40);

head= insertion(head,50);

display(head);

head = DB(head);

display(head);

head = DE(head);

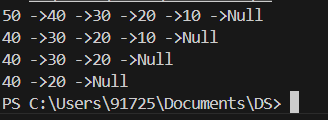
display(head);

head = DP(head,2);

display(head);

}

**Output:**



**Leet code-1 (**[**Remove Duplicates from Sorted List**](https://leetcode.com/problems/remove-duplicates-from-sorted-list/)**)**

struct ListNode\* deleteDuplicates(struct ListNode\* head) {

    struct ListNode \*q1, \*q2;

    if (head==NULL||head->next==NULL){

        return head;

    }

    q1=head;

    q2=q1->next;

    while(q2!=NULL){

        if (q1->val==q2->val){

            q1->next=q2->next;

            free(q2);

            q2=q1->next;

        }

        else{

            q1=q1->next;

            q2=q2->next;

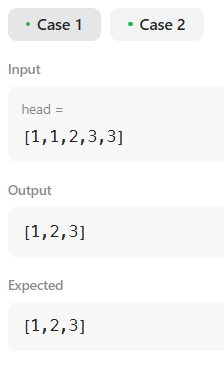
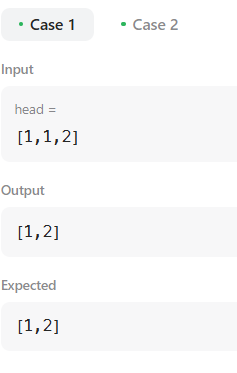
        }

    }

    return head;

}

**Output:**

****

**Leet code-2 (**[**Linked List Cycle**](https://leetcode.com/problems/linked-list-cycle/)**)**

bool hasCycle(struct ListNode \*head) {

if(head==NULL)

    return false;

  if(head-> next == NULL)

        return false;

   struct  ListNode\* slow = head;

   struct  ListNode\* fast = head;

     while(slow != NULL && fast != NULL){

         fast = fast->next;

         if(fast != NULL)

             fast = fast->next;

         slow = slow-> next;

         if(slow == fast)

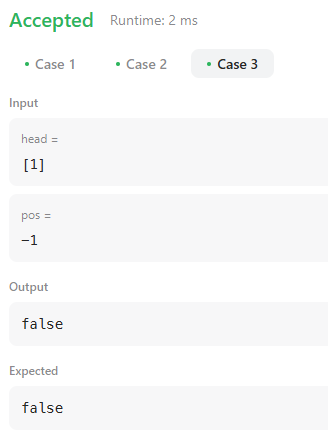
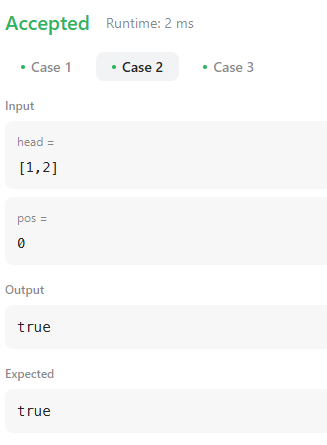
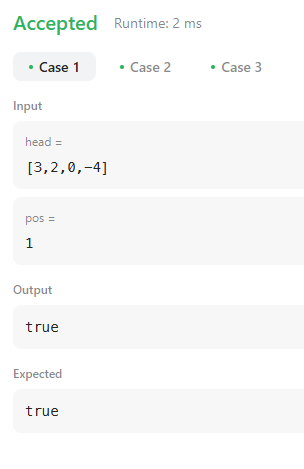
             return true;

     }

    return false;

}

**Output:**

****

**Lab program 7:**

**7a) 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, \*j;

int temp;

for (i = head; i != NULL; i = i->next) {

for (j = i->next; j != NULL; j = j->next) {

if (i->data > j->data) {

temp = i->data;

i->data = j->data;

j->data = temp;

}

}

}

}

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;

append(&list1, 3);

append(&list1, 1);

append(&list1, 2);

append(&list2, 7);

append(&list2, 6);

append(&list2, 8);

printf("List 1:\n");

printList(list1);

printf("List 2:\n");

printList(list2);

sortList(list1);

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

printList(list1);

reverseList(&list1);

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

printList(list1);

concatenateLists(&list1, list2);

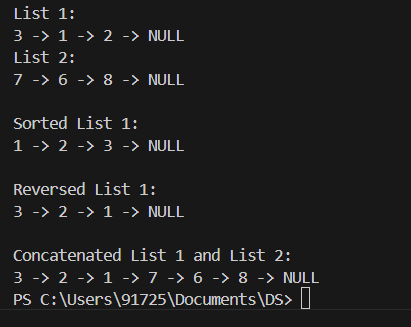
printf("\nConcatenated List 1 and List 2:\n");

printList(list1);

return 0;

}

**Output:**

****

**7b) 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;

}

struct Node \* push(struct Node\* head, int data) {

struct Node\* newNode = createNode(data);

newNode->next = head;

head = newNode;

printf("Node %d inserted.\n", data);

return head;

}

struct Node\* pop(struct Node \* head){

struct Node\* ptr;

if (head==NULL){

printf("Empty");

}

else{

ptr=head;

head=head->next;

free(ptr);

}

printf("Node deleted.\n", head->data);

return head;

}

void displayList(struct Node\* head) {

if (head == NULL) {

printf("The list is empty.\n");

}

struct Node\* temp = head;

printf("List: ");

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

}

int main() {

struct Node\* head = NULL;

int choice, data;

while (1) {

printf("\nStack:\n");

printf("\nMenu:\n");

printf("1. Push\n");

printf("2. Pop\n");

printf("3. Display \n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

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

scanf("%d", &data);

head=push(head, data);

break;

case 2:

head=pop(head);

break;

case 3:

displayList(head);

break;

case 6:

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

exit(0);

break;

default:

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

}

}

return 0;

}

#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;

}

struct Node\* insert(struct Node\* head, int data) {

struct Node\* newNode = createNode(data);

if (head == NULL) {

head = newNode;

}

else{

struct Node\* temp = head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

}

printf("Node %d inserted \n", data);

return head;

}

struct Node\* deletion(struct Node \* head){

struct Node\* ptr;

if (head==NULL){

printf("Empty");

}

else{

ptr=head;

head=head->next;

free(ptr);

}

printf("Node deleted.\n", head->data);

return head;

}

void displayList(struct Node\* head) {

if (head == NULL) {

printf("The list is empty.\n");

}

struct Node\* temp = head;

printf("List: ");

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

}

int main() {

struct Node\* head = NULL;

int choice, data;

while (1) {

printf("\nQueue:\n");

printf("\nMenu:\n");

printf("1. insertion to queue\n");

printf("2. Deletion to queue\n");

printf("3. Display \n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

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

scanf("%d", &data);

head=insert(head, data);

break;

case 2:

head=deletion(head);

break;

case 3:

displayList(head);

break;

case 6:

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

exit(0);

break;

default:

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

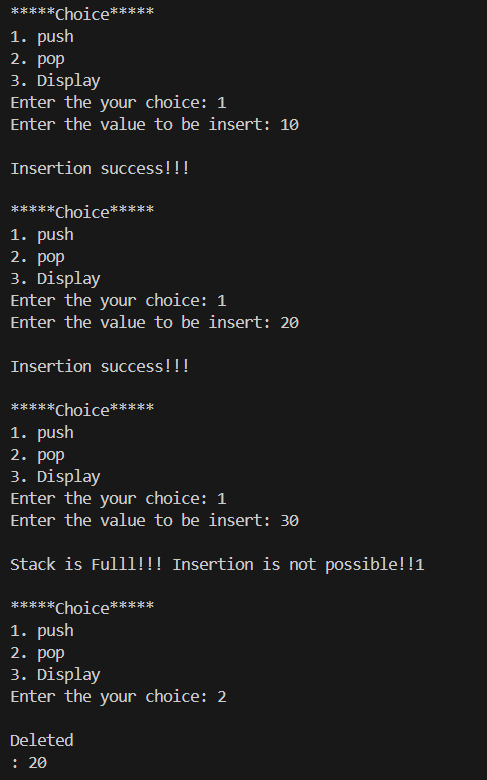
}

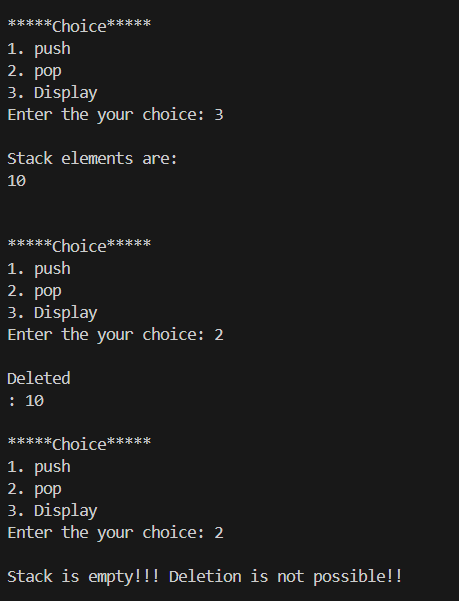
}

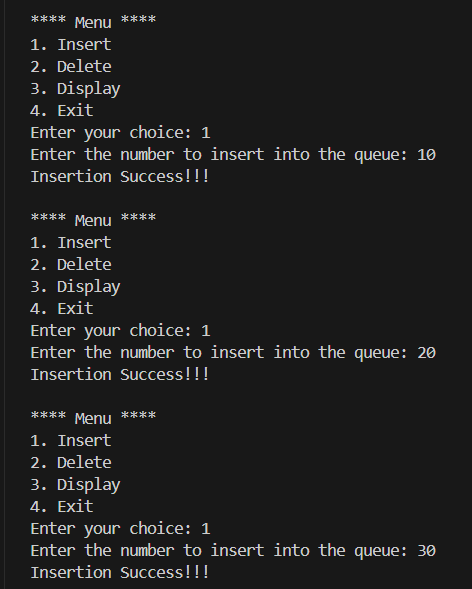
return 0;

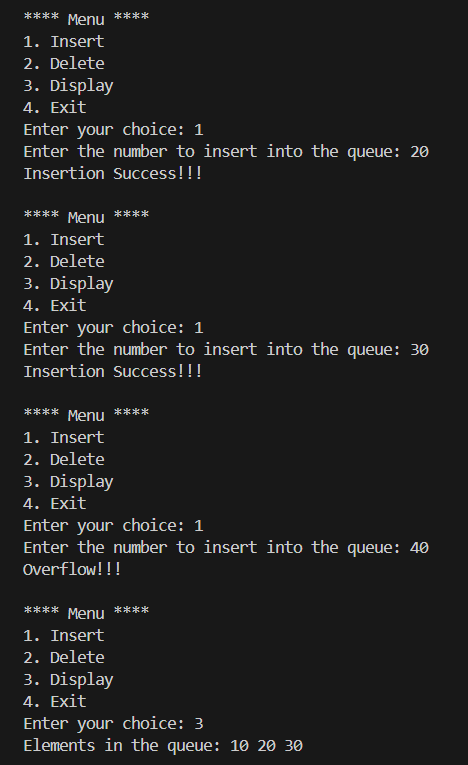
}

**Output:**

****

****

****

****

**Lab program 8:**

**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\* create\_node(int data) {

struct node\* new\_node = (struct node\*)malloc(sizeof(struct node));

new\_node->data = data;

new\_node->prev = NULL;

new\_node->next = NULL;

return new\_node;

}

struct node\* insert\_at\_position(struct node\* head, int data, int pos) {

struct node\* new\_node = create\_node(data);

if (head == NULL || pos == 0) {

new\_node->next = head;

if (head != NULL) {

head->prev = new\_node;

}

head = new\_node;

} else {

struct node\* temp = head;

int i;

for (i = 0; temp != NULL && i < pos - 1; i++) {

temp = temp->next;

}

if (temp == NULL) {

free(new\_node);

printf("Position exceeds the length of the list.\n");

return head;

} else {

new\_node->next = temp->next;

if (temp->next != NULL) {

temp->next->prev = new\_node;

}

temp->next = new\_node;

new\_node->prev = temp;

}

}

return head;

}

struct node\* delete(struct node\* head, int value) {

struct node\* temp = head;

if (head == NULL) {

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

return head;

}

if (head->data == value) {

head = head->next;

if (head != NULL) {

head->prev = NULL;

}

free(temp);

return head;

}

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

temp = temp->next;

}

if (temp == NULL) {

printf("Value not found in the list.\n");

return head;

}

if (temp->next != NULL) {

temp->next->prev = temp->prev;

}

if (temp->prev != NULL) {

temp->prev->next = temp->next;

}

free(temp);

return head;

}

void display(struct node\* head) {

struct node\* temp = head;

if (temp == NULL) {

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

return;

}

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

}

int main() {

struct node\* head = NULL;

int value, pos, choice;

while (1) {

printf("\n\*\*\* Menu \*\*\*\n");

printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value: ");

scanf("%d", &value);

printf("Enter position: ");

scanf("%d", &pos);

head = insert\_at\_position(head, value, pos);

break;

case 2:

printf("Enter value to delete: ");

scanf("%d", &value);

head = delete(head, value);

break;

case 3:

display(head);

break;

case 4:

exit(0);

default:

printf("Invalid choice. Try again.\n");

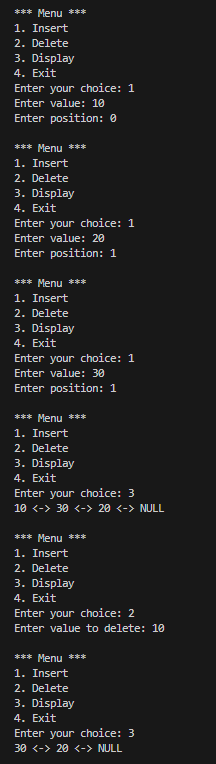
}

}

return 0;

}

**Output:**

****

**Lab program 9:**

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

#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 = newNode->right = NULL;

return newNode;

}

struct Node\* insert(struct Node\* root, int data) {

if (root == NULL)

return createNode(data);

if (data < root->data)

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

else if (data > root->data)

root->right = insert(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;

root = insert(root, 50);

insert(root, 30);

insert(root, 20);

insert(root, 40);

insert(root, 70);

insert(root, 60);

insert(root, 80);

printf("In-order Traversal: ");

inorderTraversal(root);

printf("\n");

printf("Pre-order Traversal: ");

preorderTraversal(root);

printf("\n");

printf("Post-order Traversal: ");

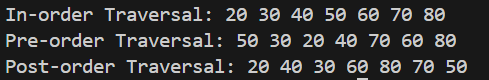
postorderTraversal(root);

printf("\n");

return 0;

}

**Output:**

****