**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**

****

**LAB REPORT**

**on**

Data Structures (23CS3PCDST)

***Submitted by***

**BHOOMIKA.M (1BM23CS068)**

***in partial fulfilment 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**

**December-2023 to Feb-2024**

**B. M. S. College of Engineering,**

**Bull Temple Road, Bangalore 560019**

(Affiliated To Visvesvaraya Technological University, Belgaum)

**Department of Computer Science and Engineering**

**CERTIFICATE**

This is to certify that the Lab work entitled “Data Structures (23CS3PCDST)” carried out by who BHOOMIKA.M (1BM23CS068) is a bonafide student of **B. M. S. College of Engineering.** It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2024. The Lab report has been approved as it satisfies the academic requirements in respect of a Data Structures(23CS3PCDST) work prescribed for the said degree.

Dr. Selva Kumar S

Dr. Kavitha Sooda

Associate Professor

Professor & HOD

Department of CSE, BMSCE

Department of CSE, BMSCE

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| S.NO | DATE | TITLE | PAGE NO. |
| 1. | 3/10/24 | Working of stack | 1 |
| 2. | 24/10/24 | Infix to postfix expression | 6 |
| 3. | 24/10/24 | Linear queue and circular queue | 8 |
| 4. | 7/11/24 | Singly linked list - insertion | 13 |
|  |  | Singly linked list - deletion | 16 |
| 5. | 21/11/24 | Sort,reverse,concatenate singly linked list | 19 |
| 6. | 21/11/24 | Stimulate stack and queue | 23 |
| 7. | 28/11/24 | Doubly linked list | 32 |
| 8. | 19/12/24 | Binary search tree | 36 |
| 9. | 26/12/24 | BSF, DFS | 39 |
| 10. | 26/12/24 | HASHING | 43 |
| 11. |  | Leetcode questions | 46 |

GITHUB LINK

https://github.com/Bhoomika-M-CSE/Bhoomika

**LAB PROGRAMS**

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.

Program:

#include<stdio.h> #include<conio.h> #define SIZE 5 void push(int); void pop();

void display();

int stack[SIZE],top=-1; void main()

{

int choice,value; int clrscr(); while(1)

{

printf("\nMENU\n");

printf("1.push\n 2.pop\n 3.display\n 4.exit\n"); printf("Enter your choice:"); scanf("%d",&choice);

switch(choice)

{

case 1: printf("Enter the value to be inserted: "); scanf("%d",&value);

push(value); break;

case 2:pop(); break;

case 3:display(); break;

case 4: exit(0);

default:printf("WRONG SELECTION");

}

}

}

void push(int value)

{

if(top==SIZE-1)

{

printf("Stack is full");

}

else

{

top++; stack[top]=value;

printf("Insertion successful");

}

}

void pop()

{

if(top==-1)

{

printf("Stack is empty");

}

else

{

printf("deleted=%d",stack[top]); top--;

}

}

void display()

{

if(top==-1)

{

printf("stack is empty,underflow");

}

else

{

int i;

printf("stack elements are:"); for(i=top;i>=0;i--)

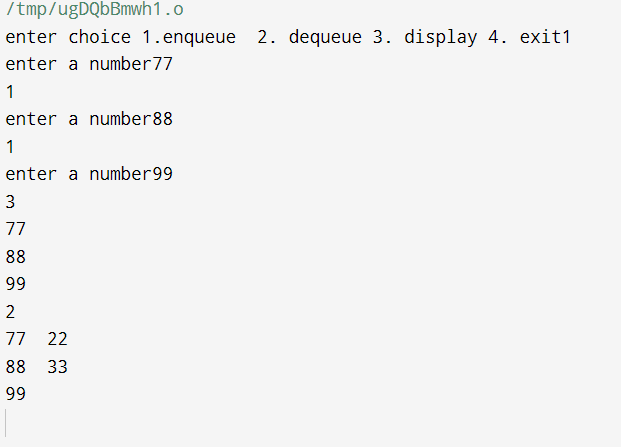
{

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

}

}

}



1. 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)

Program:

#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 postfix[50], infix[50], ch, elem; int i = 0, k = 0;

printf("Enter the infix expression: "); scanf("%s", infix);

push('#');

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

{

if (ch == '(')

{

push(ch);

}

else if (isalnum(ch))

{

postfix[k++] = ch;

}

else if (ch == ')')

{

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

{

postfix[k++] = pop();

}

pop();

}

else

{

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

{

postfix[k++] = pop();

}

push(ch);

}

}

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

{

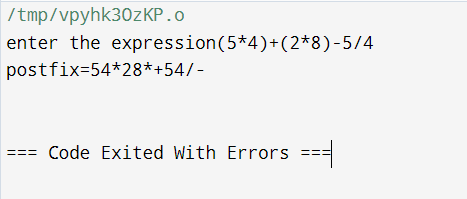
postfix[k++] = pop();

}

postfix[k] = '\0';

printf("Postfix expression = %s\n", postfix);

}



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.

Program:

#include<stdio.h> #define Max 5

int queue[Max]; int front=-1;

int rear=-1;

void insert(int item); void delete();

void display();

void main()

{

int choice, item; while(1)

{

printf("\nMENU\n"); printf("1. Insert\n"); printf("2. Delete\n"); printf("3. Display\n"); printf("4. Exit\n"); printf("Enter your choice: "); scanf("%d", &choice); switch(choice)

{

case 1:

printf("Enter the element to insert: "); scanf("%d", &item);

insert(item); break;

case 2:

delete(); break;

case 3:

display(); break;

case 4:

exit(0); default:

printf("Invalid choice\n");

}

}

}

void insert(int add\_item)

{

if(rear == Max-1)

{

printf("Queue overflow\n");

}

else

{

if(front == -1)

{

front = 0;

}

rear = rear + 1; queue[rear] = add\_item;

printf("Inserted %d\n", add\_item);

}

}

void delete()

{

if(front == -1 || front > rear)

{

printf("Queue underflow\n"); return;

}

else

{

printf("Deleted item is %d\n", queue[front]); front = front + 1;

}

}

void display()

{

int i;

if(front == -1)

{

printf("Queue is empty\n");

}

else

{

printf("Queue is: ");

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

{

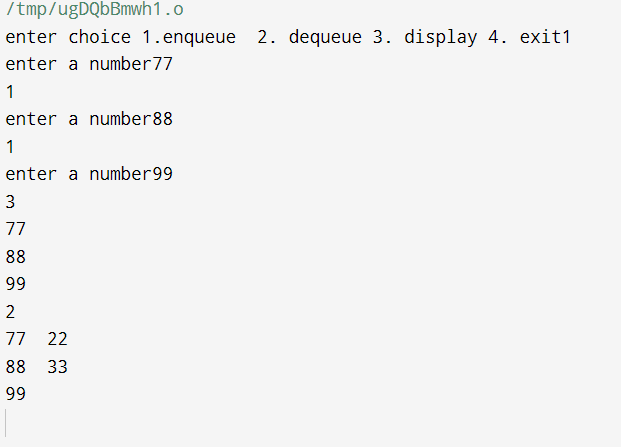
printf("%d ", queue[i]);

}

printf("\n");

}

}



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.

Program:

#include<stdio.h> #define Max 5

int queue[Max];

int front = -1;

int rear = -1;

void insert(int item); void delete();

void display(); void main() {

int choice, item; while(1) {

printf("\nMENU\n"); printf("1. Insert\n"); printf("2. Delete\n"); printf("3. Display\n"); printf("4. Exit\n"); printf("Enter your choice: "); scanf("%d", &choice);

switch(choice) {

case 1:

printf("Enter the element to insert: "); scanf("%d", &item);

insert(item); break;

case 2:

delete(); break;

case 3:

display(); break;

case 4:

exit(0); default:

printf("Invalid choice\n");

}

}

}

void insert(int item)

{

if ((front == 0 && rear == Max - 1) || (rear == (front - 1) % (Max - 1)))

{

printf("Queue overflow\n"); return;

}

else if (front == -1)

{

front = rear = 0; queue[rear] = item;

}

else if (rear == Max - 1 && front != 0)

{

rear = 0; queue[rear] = item;

}

else

{

rear++; queue[rear] = item;

}

printf("Inserted %d\n", item);

}

void delete()

{

if (front == -1) {

printf("Queue underflow\n"); return;

}

printf("Deleted item is %d\n", queue[front]); if (front == rear)

{

front = rear = -1;

}

else if (front == Max - 1)

{

front = 0;

}

else

{

front++;

}

}

void display() { int i;

if (front == -1) { printf("Queue is empty\n"); return;

}

printf("Queue is: "); if (rear >= front)

{

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

{

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

}

}

else

{

for(i = front; i < Max; i++)

{

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

}

for(i = 0; i <= rear; i++)

{

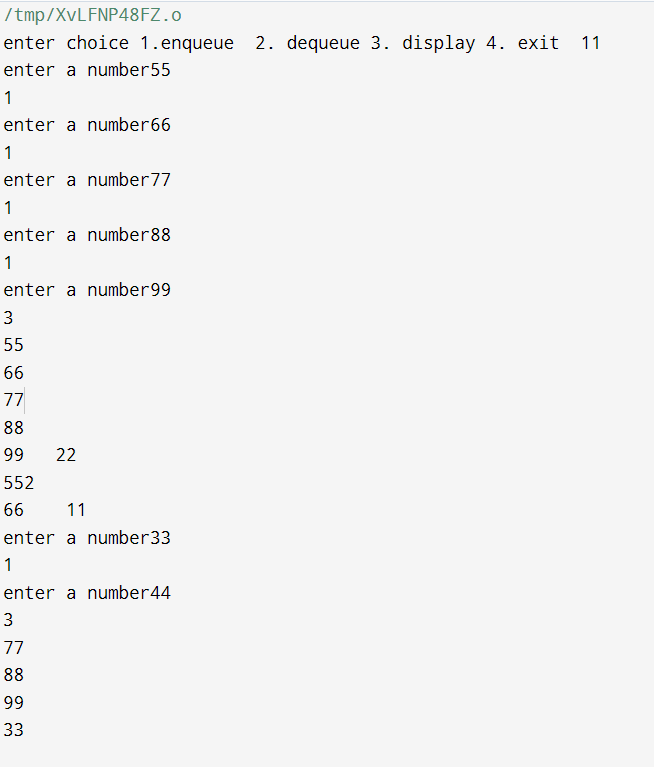
printf("%d ", queue[i]);

}

}

printf("\n");

}



1. WAP to Implement Singly Linked List with following operations a) Createalinkedlist. b) Insertion of a node at first position, at any position and at end of list. Display the contents of the linked list.

Program:

#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 insertAtFirst(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\* temp = \*head; while (temp->next != NULL)

{

temp = temp->next;

}

temp->next = newNode;

}

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

{

struct Node\* newNode = createNode(data); if (position == 0)

{

insertAtFirst(head,data); return;

}

struct Node\* temp = \*head;

for (int i = 0; temp != NULL && i < position - 1; i++)

{

temp = temp->next;

}

if (temp == NULL)

{

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

return;

}

newNode->next = temp->next; temp->next = newNode;

}

void display(struct Node\* head)

{

struct Node\* temp = head; while (temp != NULL)

{

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

}

printf("NULL\n");

}

int main()

{

struct Node\* head = NULL;

printf("Linked list after inserting the node:10 at the beginning \n"); insertAtFirst(&head, 10);

display(head);

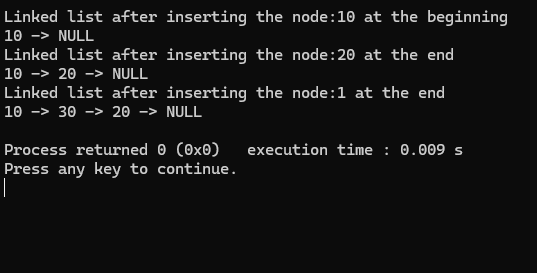
printf("Linked list after inserting the node:20 at the end \n"); insertAtEnd(&head, 20);

display(head);

printf("Linked list after inserting the node:1 at the end \n"); insertAtPosition(&head,30,1);

display(head);

}



1. 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.

Program:

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

struct node { int value;

struct node\* next;

};

typedef struct node\* NODE; NODE get\_node() {

NODE ptr = (NODE)malloc(sizeof(struct node)); if (ptr == NULL) {

printf("Memory not allocated\n");

}

return ptr;

}

NODE delete\_first(NODE first) { NODE temp = first;

if (first == NULL) { printf("Linked list is empty\n"); return NULL;

}

first = first->next; free(temp);

return first;

}

NODE delete\_last(NODE first) { NODE prev, last;

if (first == NULL) { printf("Linked list is empty\n"); return NULL;

}

prev = NULL; last = first;

while (last->next != NULL)

{

prev = last;

last = last->next;

}

if (prev == NULL)

{

free(first); return NULL;

}

prev->next = NULL; free(last);

return first;

}

NODE delete\_value(NODE first, int value\_del) { if (first == NULL) {

printf("Linked list is empty\n"); return NULL;

}

NODE prev = NULL;

NODE current = first;

while (current != NULL && current->value != value\_del) { prev = current;

current = current->next;

}

if (current == NULL) { printf("Value not found\n"); return first;

}

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

} else {

prev->next = current->next;

}

free(current); return first;

}

void display(NODE first) { NODE temp = first;

if (first == NULL) { printf("Empty\n"); return;

}

while (temp != NULL) { printf("%d ", temp->value); temp = temp->next;

}

printf("\n");

}

NODE insert\_beginning(NODE first, int item) { NODE new\_node = get\_node();

new\_node->value = item; new\_node->next = first; return new\_node;

}

int main() {

NODE head = NULL;

int choice, item;

head = insert\_beginning(head, 1); head = insert\_beginning(head, 2); head = insert\_beginning(head, 3); head = insert\_beginning(head, 4);

while (1) {

printf("1. Delete first\n"); printf("2. Delete last\n"); printf("3. Delete value\n"); printf("4. Display\n"); printf("5. Exit\n"); printf("Enter your choice: "); scanf("%d", &choice);

switch (choice) { case 1:

head = delete\_first(head); break;

case 2:

head = delete\_last(head); break;

case 3:

printf("Enter value to delete: "); scanf("%d", &item);

head = delete\_value(head, item); break;

case 4:

display(head); break;

case 5:

return 0; default:

printf("Invalid choice\n");

}

}

return 0;

}



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

Program:

#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 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;

}

}

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; struct Node\* current = \*head; struct Node\* 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 choice, data;

temp = i->data;

i->data = j->data;

j->data = temp;

}

}

}

}

void reverseList(struct Node\*\* head) {

struct Node\* prev = NULL; struct Node\* current = \*head; struct Node\* 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 choice, data;

while (1) {

printf("\n1. Insert into List 1\n"); printf("2. Insert into List 2\n"); printf("3. Sort List 1\n"); printf("4. Reverse List 1\n");

printf("5. Concatenate List 1 and List 2\n"); printf("6. Print List 1\n");

printf("7. Print List 2\n"); printf("8. Exit\n"); printf("Enter your choice: "); scanf("%d", &choice);

switch (choice) { case 1:

printf("Enter data to insert into List 1: "); scanf("%d", &data);

insert(&list1, data); break;

case 2:

printf("Enter data to insert into List 2: "); scanf("%d", &data);

insert(&list2, data); break;

case 3:

sortList(list1); printf("List 1 sorted.\n"); break;

case 4:

reverseList(&list1); printf("List 1 reversed.\n"); break;

case 5:

concatenateLists(&list1, list2); printf("List 2 concatenated to List 1.\n"); break;

case 6:

printf("List 1: "); printList(list1); break;

case 7:

printf("List 2: "); printList(list2); break;

case 8:

exit(0); default:

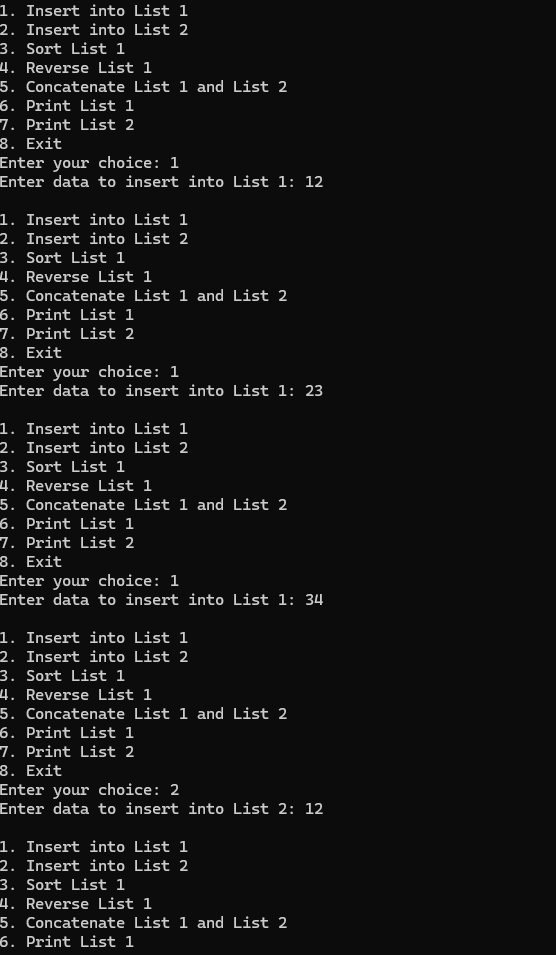
printf("Invalid choice! Please try again.\n");

}

}

return 0;

}



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

Program:

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

struct node { int value;

struct node \*next;

};

typedef struct node \*NODE;

NODE get\_node() {

NODE ptr = (NODE)malloc(sizeof(struct node)); if (ptr == NULL) {

printf("Memory not allocated\n");

}

return ptr;

}

NODE delete\_first(NODE first){ NODE temp=first;

if (first == NULL) { printf("Empty\n"); return NULL;

}

first=first->next; free(temp); return first;

}

NODE insert\_beginning(NODE first, int item) { NODE new\_node = get\_node();

new\_node->value = item; new\_node->next = first; return new\_node;

}

NODE insert\_end(NODE first, int item) { NODE new\_node = get\_node(); new\_node->value = item;

new\_node->next = NULL; if (first == NULL) {

return new\_node;

}

NODE temp = first;

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

}

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

}

void display(NODE first) { NODE temp = first;

if (first == NULL) { printf("Empty\n"); return;

}

while (temp != NULL) { printf("%d ", temp->value); temp = temp->next;

}

printf("\n");

}

int main() {

int item, choice, deleted\_item; NODE first = NULL;

printf("Choose:\n"); printf("1. Stack\n"); printf("2. Queue\n"); printf("Enter choice (1/2): "); scanf("%d", &choice);

if (choice == 1) { while (1) {

printf("\nStack Operations:\n"); printf("1. Push\n");

printf("2. Pop\n"); printf("3. Display stack\n");

printf("4. Exit\n"); printf("Enter choice: "); scanf("%d", &choice);

switch (choice) { case 1:

printf("Enter item to push: "); scanf("%d", &item);

first = insert\_beginning(first, item); break;

case 2:

if (first != NULL) { deleted\_item = first->value; first = delete\_first(first);

printf("Deleted item from stack: %d\n", deleted\_item);

} else {

printf("Stack is empty\n");

}

Break;

; case 3:

printf("Stack: "); display(first); break;

case 4:

exit(0); default:

printf("Invalid choice.\n");

}

}

}

else if (choice == 2) { while (1) {

printf("\nQueue Operations:\n"); printf("1. Insert\n");

printf("2. Delete\n"); printf("3. Display queue\n"); printf("4. Exit\n"); printf("Enter choice: "); scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter item to insert: "); scanf("%d", &item);

first = insert\_end(first, item); break;

case 2:

if (first != NULL) { deleted\_item = first->value; first = delete\_first(first);

printf("Deleted item from queue: %d\n", deleted\_item);

} else {

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

}

break; case 3:

printf("Queue: "); display(first); break;

case 4:

exit(0); default:

printf("Invalid choice.\n");

}

}

}

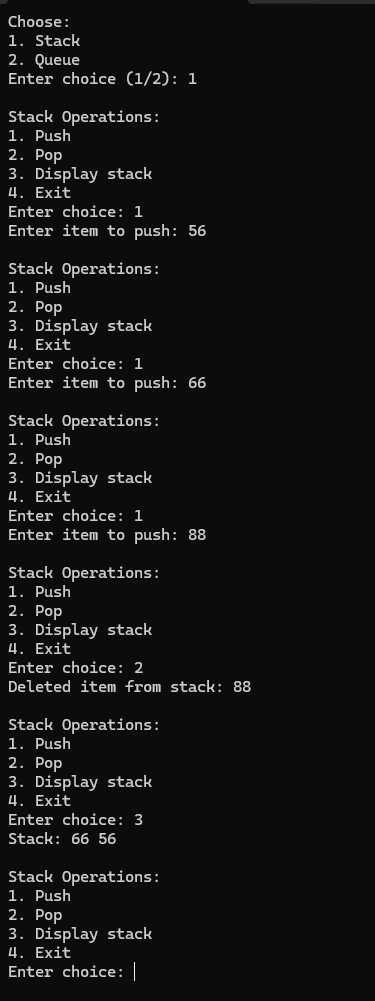
else {

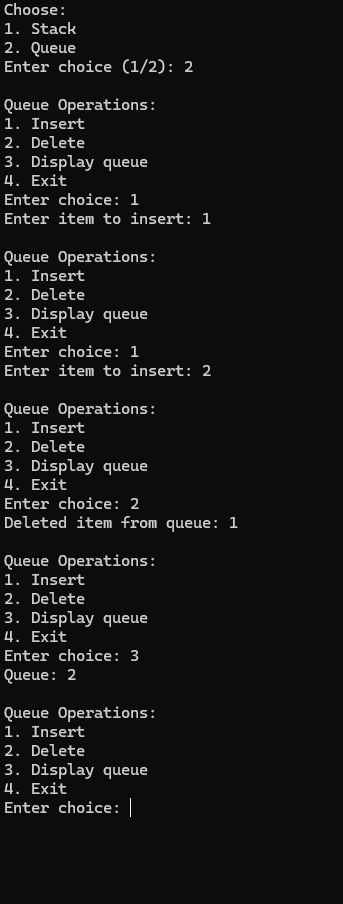
printf("Invalid operation.\n");

}

return 0;

}





1. 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

Program:

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

{

int data;

struct Node\* prev; struct Node\* next;

};

void create(struct Node\*\* head, int data)

{

struct Node\* new\_node = (struct Node\*)malloc(sizeof(struct Node)); new\_node->data = data;

new\_node->prev = NULL; new\_node->next = NULL; if (\*head == NULL)

{

\*head = new\_node; return;

}

struct Node\* temp = \*head; while (temp->next != NULL)

{

temp = temp->next;

}

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

}

void insert\_left(struct Node\*\* head, int target\_data, int new\_data)

{

struct Node\* new\_node = (struct Node\*)malloc(sizeof(struct Node)); new\_node->data = new\_data;

struct Node\* temp = \*head; while (temp != NULL)

{

if (temp->data == target\_data)

{

new\_node->next = temp; new\_node->prev = temp->prev; if (temp->prev != NULL)

{

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

}

else

{

\*head = new\_node;

}

temp->prev = new\_node; return;

}

temp = temp->next;

}

printf("Node with data %d not found.\n", target\_data);

}

void delete\_node(struct Node\*\* head, int value)

{

struct Node\* temp = \*head; while (temp != NULL)

{

if (temp->data == value)

{

if (temp == \*head)

{

\*head = temp->next;

}

if (temp->prev != NULL)

{

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

}

if (temp->next != NULL)

{

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

}

free(temp); return;

}

temp = temp->next;

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

}

void display(struct Node\* head)

{

if (head == NULL) { printf("The list is empty.\n"); return;

}

}

struct Node\* temp = head; while (temp != NULL)

{

printf("%d", temp->data); if (temp->next != NULL)

{

printf(" <-> ");

}

temp = temp->next;

}

printf("\n");

}

int main()

{

struct Node\* head = NULL;

int choice, data, target\_data, new\_data;

while (1)

{

printf("\nDoubly Linked List Operations:\n"); printf("1. Create a node\n");

printf("2. Insert node to the left of a specific node\n"); printf("3. Delete a node\n");

printf("4. Display the list\n"); printf("5. Exit\n"); printf("Enter your choice: "); scanf("%d", &choice);

switch (choice)

{

case 1:

printf("Enter the data for the node to create: ");

scanf("%d", &data); create(&head, data); break;

case 2:

printf("Enter the target node data before which to insert: "); scanf("%d", &target\_data);

printf("Enter the data for the new node to insert: "); scanf("%d", &new\_data);

insert\_left(&head, target\_data, new\_data); break;

case 3:

printf("Enter the data of the node to delete: "); scanf("%d", &data);

delete\_node(&head, data); break;

case 4:

printf("The current list is: "); display(head);

break;

case 5:

printf("Exiting...\n"); exit(0);

default:

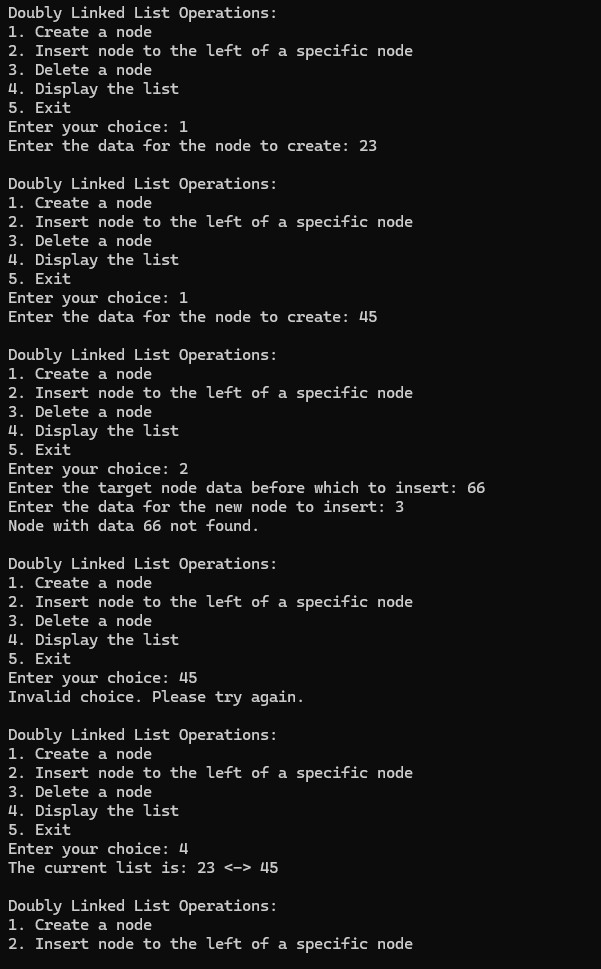
printf("Invalid choice. Please try again.\n");

}

}

return 0;

}



1. Write a program a) ToconstructabinarySearchtree. b) To traverse the tree using all the methods i.e., inorder, preorder and post order c) To display the elements in the tree.

Program:

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

{

int data;

struct node \*left; struct node \*right;

};

struct node\* newNode(int data)

{

struct node\* node = (struct node\*)malloc(sizeof(struct node)); node->data = data;

node->left = node->right = NULL; return node;

}

struct node\* insert(struct node\* root, int data)

{

if (root == NULL)

return newNode(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 inorder(struct node\* root)

{

if (root != NULL)

{

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

}

}

void preorder(struct node\* root)

{

if (root != NULL)

{

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

}

}

void postorder(struct node\* root)

{

if (root != NULL)

{

postorder(root->left); postorder(root->right);

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

}

}

void display(struct node\* root, int choice)

{

switch (choice)

{

case 1:

printf("\nIn-order traversal: "); inorder(root);

break; case 2:

printf("\nPre-order traversal: "); preorder(root);

break; case 3:

printf("\nPost-order traversal: "); postorder(root);

break; default:

printf("\nInvalid choice\n"); break;

}

}

int main()

{

struct node\* root = NULL; int n, data, choice;

printf("Enter the number of nodes to insert in the BST: "); scanf("%d", &n);

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

{

printf("Enter value for node %d: ", i + 1); scanf("%d", &data);

root = insert(root, data);

}

while (1)

{

printf("\nChoose the type of traversal:\n"); printf("1. In-order\n");

printf("2. Pre-order\n"); printf("3. Post-order\n"); printf("4. Exit\n");

printf("Enter your choice (1/2/3/4): "); scanf("%d", &choice);

if (choice == 4)

{

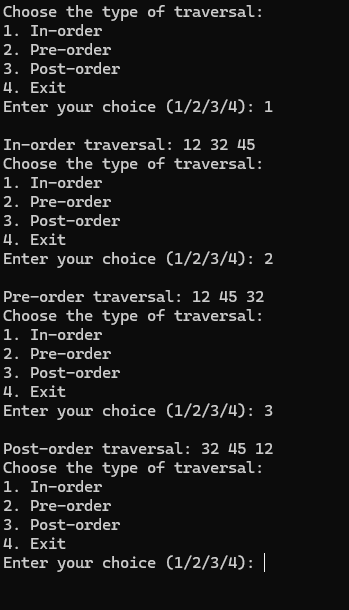
printf("Exiting the program...\n"); break;

}

display(root, choice);

}

return 0;

}

1. a) Write a program to traverse a graph using BFS method. Program:

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

#define MAX 100 struct Queue {

int items[MAX];

int front, rear;

};

void initQueue(struct Queue\* q) { q->front = -1;

q->rear = -1;

}

bool isEmpty(struct Queue\* q) { return q->front == -1;

}

void enqueue(struct Queue\* q, int value) { if (q->rear == MAX - 1)

return;

if (q->front == -1) q->front = 0;

q->rear++;

q->items[q->rear] = value;

}

int dequeue(struct Queue\* q) { if (isEmpty(q))

return -1;

int item = q->items[q->front]; if (q->front == q->rear) {

q->front = q->rear = -1;

} else {

q->front++;

}

return item;

}

struct Graph { int vertices;

int adjMatrix[MAX][MAX];

};

void initGraph(struct Graph\* g, int vertices) { g->vertices = vertices;

for (int i = 0; i < vertices; i++) { for (int j = 0; j < vertices; j++) {

g->adjMatrix[i][j] = 0;

}

}

}

void addEdge(struct Graph\* g, int u, int v) { g->adjMatrix[u][v] = 1;

g->adjMatrix[v][u] = 1;

}

void bfs(struct Graph\* g, int start) { bool visited[MAX] = {false}; struct Queue q;

initQueue(&q); visited[start] = true; enqueue(&q, start);

while (!isEmpty(&q)) {

int node = dequeue(&q); printf("%d ", node);

for (int i = 0; i < g->vertices; i++) {

if (g->adjMatrix[node][i] == 1 && !visited[i]) { visited[i] = true;

enqueue(&q, i);

}

}

}

}

int main() { struct Graph g;

initGraph(&g, 6);

addEdge(&g, 0, 1);

addEdge(&g, 0, 2);

addEdge(&g, 1, 3);

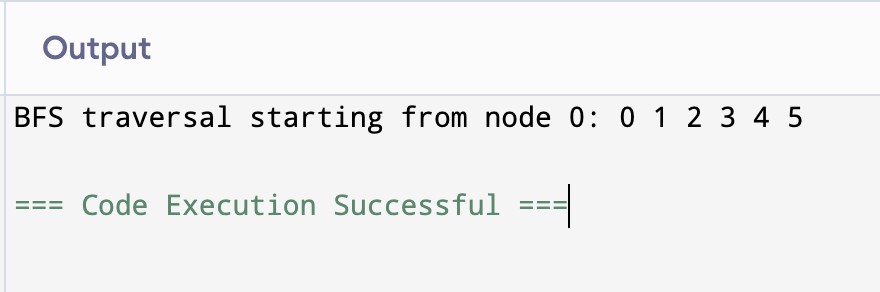
addEdge(&g, 1, 4);

addEdge(&g, 2, 5);

printf("BFS traversal starting from node 0: "); bfs(&g, 0);

return 0;

}



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

Program:

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

#define MAX 100 struct Graph {

int vertices;

int adjMatrix[MAX][MAX];

};

void initGraph(struct Graph\* g, int vertices) { g->vertices = vertices;

for (int i = 0; i < vertices; i++) { for (int j = 0; j < vertices; j++) {

g->adjMatrix[i][j] = 0;

}

}

}

void addEdge(struct Graph\* g, int u, int v) { g->adjMatrix[u][v] = 1;

g->adjMatrix[v][u] = 1;

}

void dfs(struct Graph\* g, int vertex, bool visited[]) { visited[vertex] = true;

for (int i = 0; i< g->vertices; i++) {

if (g->adjMatrix[vertex][i] == 1 && !visited[i]) { dfs(g, i, visited);

}

}

}

bool isConnected(struct Graph\* g) { bool visited[MAX] = {false}; dfs(g, 0, visited);

for (int i = 0; i < g->vertices; i++) { if (!visited[i]) {

return false;

}

}

return true;

}

int main() { struct Graph g;

int vertices = 6; initGraph(&g, vertices);

addEdge(&g, 0, 1);

addEdge(&g, 0, 2);

addEdge(&g, 1, 3);

addEdge(&g, 1, 4);

addEdge(&g, 2, 5);

if (isConnected(&g)) {

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

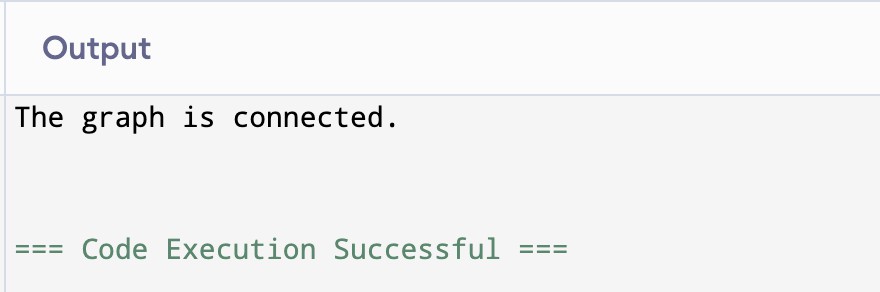
} else {

printf("The graph is not connected.\n");

}

return 0;

}



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 -&gt; L as

H(K)=K mod 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>

#include <string.h>

#define MAX 100

#define M 10

typedef struct {

int key,

char name[50];

char department[30];

} Employee;

typedef struct {

int key;

Employee emp;

int isOccupied;

} HashTableEntry;

int hashFunction(int key) {

return key % M;

}

void insert(HashTableEntry hashTable[], Employee emp) {

int index = hashFunction(emp.key);

int originalIndex = index;

while (hashTable[index].isOccupied) {

if (hashTable[index].key == emp.key) {

printf("Error: Duplicate key detected!\n");

return;

}

index = (index + 1) % M;

if (index == originalIndex) {

printf("Error: Hash table is full!\n");

return;

}

}

hashTable[index].key = emp.key;

hashTable[index].emp = emp;

hashTable[index].isOccupied = 1;

printf("Inserted key %d at index %d\n", emp.key, index);

}

Employee \*search(HashTableEntry hashTable[], int key) {

int index = hashFunction(key);

int originalIndex = index;

while (hashTable[index].isOccupied) {

if (hashTable[index].key == key) {

return &hashTable[index].emp;

}

index = (index + 1) % M;

if (index == originalIndex) {

break;

}

}

return NULL;

}

void displayHashTable(HashTableEntry hashTable[]) {

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

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

if (hashTable[i].isOccupied) {

printf("Index %d: Key = %d, Name = %s, Department = %s\n",

i, hashTable[i].key, hashTable[i].emp.name, hashTable[i].emp.department);

} else {

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

}

}

}

int main() {

HashTableEntry hashTable[M];

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

hashTable[i].isOccupied = 0;

}

int choice;

Employee emp;

do {

printf("\nMenu:\n");

printf("1. Insert Employee Record\n");

printf("2. Search Employee Record\n");

printf("3. Display Hash Table\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter 4-digit key: ");

scanf("%d", &emp.key);

printf("Enter name: ");

scanf("%s", emp.name);

printf("Enter department: ");

scanf("%s", emp.department);

insert(hashTable, emp);

break;

case 2:

printf("Enter key to search: ");

int key;

scanf("%d", &key);

Employee \*result = search(hashTable, key);

if (result) {

printf("Employee Found: Key = %d, Name = %s, Department = %s\n",

result->key, result->name, result->department);

} else {

printf("Employee with key %d not found.\n", key);

}

break;

case 3:

displayHashTable(hashTable);

break;

case 4:

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

break;

default:

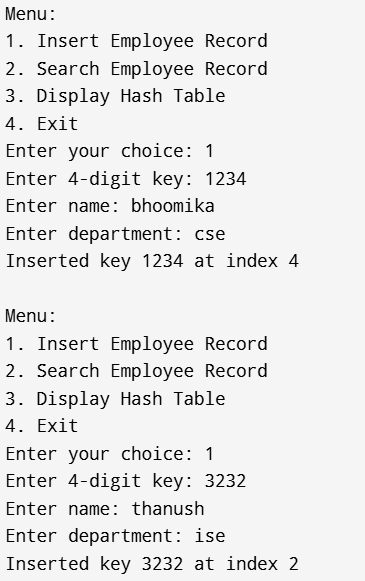
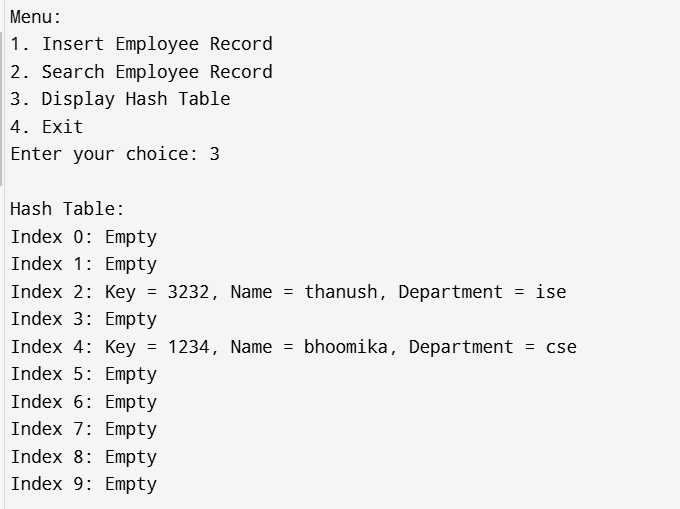
printf("Invalid choice!\n");

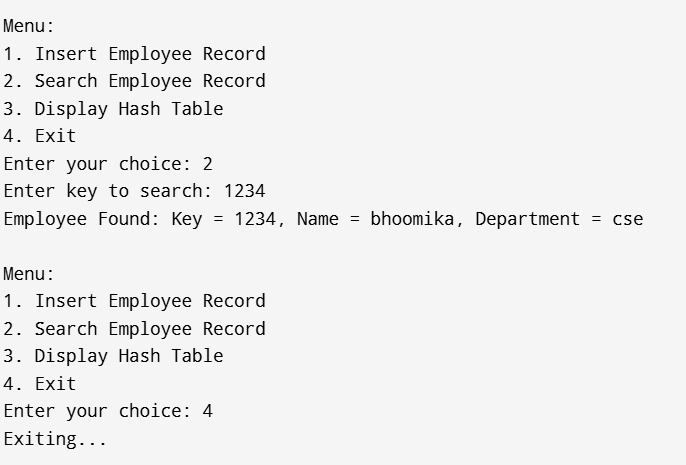
}

} while (choice != 4);

return 0;

}



**LEET CODE QUESTIONS**

1. **Move zeros**

void moveZeroes(int\* nums, int numsSize) {

for(int i=0; i<numsSize-1; i++){

if(nums[i]==0){

for(int k=i; k<numsSize-1; k++){

int temp=nums[k];

nums[k]=nums[k+1];

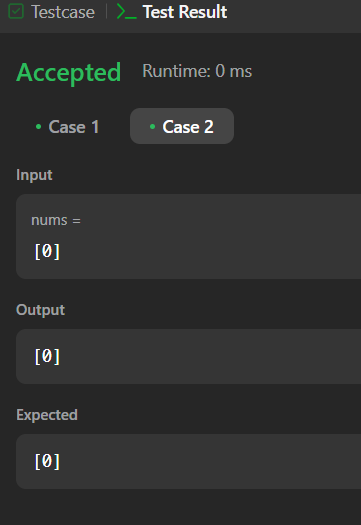
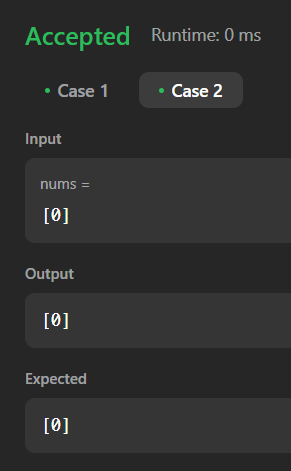
nums[k+1]=temp;

}

}

}

}

1. **Implement Stack using Queues**

typedef struct {

int \*obj;

int size;

int rear;

int front;

} MyStack;

MyStack\* myStackCreate() {

MyStack \*obj=(MyStack\*)malloc(sizeof(MyStack));

int front=-1;

int rear=-1;

return obj;

}

void myStackPush(MyStack\* obj, int x) {

if (obj->front==-1) obj->front=obj->rear=0;

else if(obj->rear<obj->size){

obj->rear=obj->rear+1;

}

obj->rear=obj->rear=x;

}

int myStackPop(MyStack\* obj) {

return obj->rear--;

}

int myStackTop(MyStack\* obj) {

return obj->rear;

}

if(obj->rear==-1)return 1;

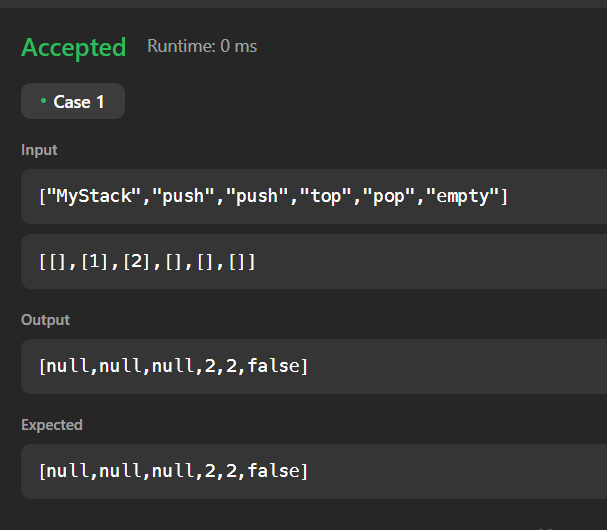
else return 0;

}

void myStackFree(MyStack\* obj) {

free(obj);

}



1. **Remove all adjacent duplicates in a string**

char\* removeDuplicates(char\* s) { int n = strlen(s);

char\* stack = malloc(sizeof(char) \* (n + 1)); int i = 0;

for (int j = 0; j < n; j++) { char c = s[j];

if (i && stack[i - 1] == c) { i--;

} else {

stack[i++] = c;

}

}

stack[i] = '\0'; return stack;

}



1. **Backspace String Compare**

typedef struct node{

char \*st;

int top;

int capacity;

}stack;

stack\* init(int capacity){

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

node->top=-1;

node->capacity=capacity;

node ->st=(char\*)malloc(capacity\*sizeof(char));

return node;

}

void push(stack\* node,char x ){

if(node->top==node->capacity-1){

printf("stack is full");

return;}

node->st[++node->top]=x;

}

char pop(stack \* node){

return node->st[node->top--];

}

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

int capacity=50;

stack \* node=init(capacity);

int r,i=0;

while(\*s!='\0'){

if(\*s=='#'){

char a;

a=pop(node);

}

else push(node,\*s);

s++;

}

char arr1[capacity];

while(node->top>=0){

arr1[i++]=pop(node);

}

free(node);

stack \* node1=init(capacity);

while(\*t!='\0'){

if(\*t=='#'){

char a;

a=pop(node1);

}

else push(node1,\*t);

t++;

}

char arr2[capacity];

int j=0;

while(node1->top>=0){

arr2[j++]=pop(node1);

}

free(node1);

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

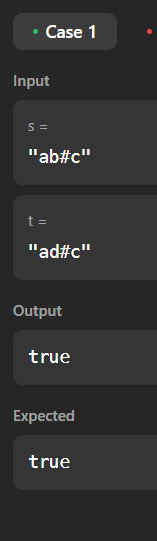
if(arr1[k]==arr2[k]) r=1;

else r=0;

}

return r;

}



1. **Remove digit from number to maximize result**

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

{

int len = strlen(number);

for (int i = 0; i < len - 1; i++)

{

if (number[i] == digit && number[i] < number[i + 1])

{

for (int j = i; j < len - 1; j++)

{

number[j] = number[j + 1];

}

number[len - 1] = '\0'; return number;

}

}

for (int i = len - 1; i >= 0; i--)

{

if (number[i] == digit)

{

for (int j = i; j < len - 1; j++)

{

number[j] = number[j + 1];

}

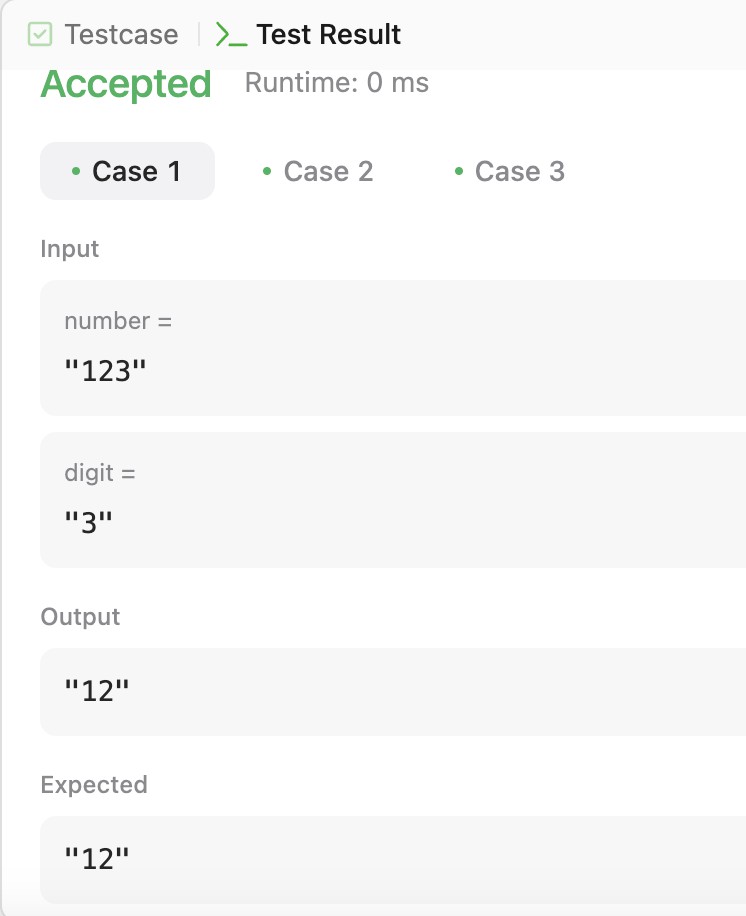
number[len - 1] = '\0'; return number;

}

}

return number;

}



1. **Remove duplicates from sorted list**

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

{

if (head == NULL) { return head;

}

struct ListNode\* current = head;

while (current != NULL && current->next != NULL)

{

if (current->val == current->next->val)

{

struct ListNode\* temp = current->next; current->next = current->next->next; free(temp);

}

else

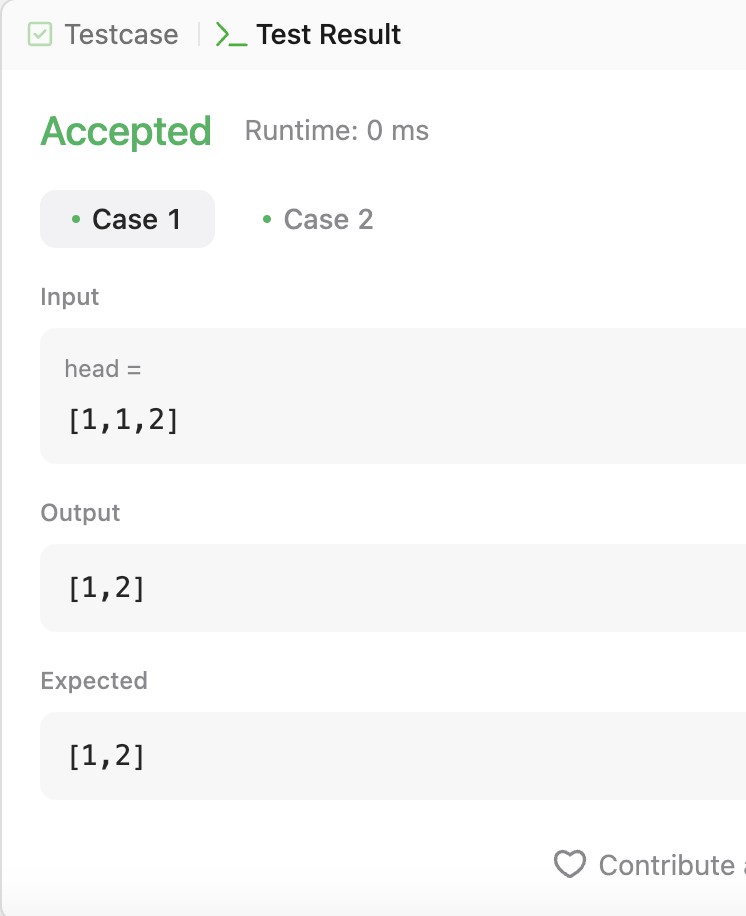
{

current = current->next;

}

}

return head;

}