DS LAB REPORT

SUHAS KUMAR REDDY NARE(1BM22CS294)

1. Write a program to simulate the working of stack using an array with the following: a) Push b) Pop c) Display the program should print appropriate messages for stack overflow, stack underflow

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

int stack[n]; int top=-1; void push(); void pop(); void display(); void push()

{

int item; if(top==n-1)

{

printf("stack is full,overflow condition"); return;

}

else

{

printf("enter the number to be inserted"); scanf("%d",&item);

top++; stack[top]=item;

}

}

void pop()

{

int item; if(top==-1)

{

printf("stack is empty,underflow condition"); return;

}

else

{

printf("enter the number to be deleted\n"); scanf("%d",&item);

item=stack[top]; top=top-1;

}

}

void display()

{

int i;

printf("elements of stack are\t"); for(i=top;i>=0;i--)

{

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

}

if(top==-1)

{

printf("stack is empty");

}

}

void main()

{

int choice;

printf("print the choices 1.push 2.pop 3.display 4.exits\n"); printf("read choice");

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

{

switch(choice)

{

case 1:push(); break;

case 2:pop(); break;

case 3:display(); break;

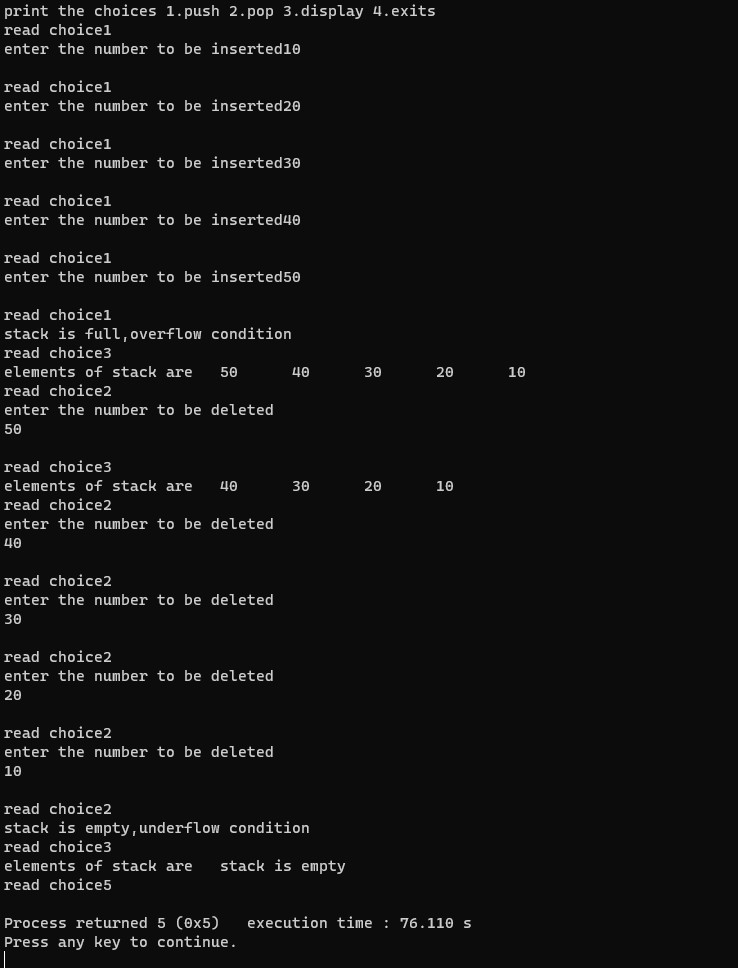
case 4:exit(0); default:printf("invalid choice"); break;

}

printf("\nread choice"); scanf("%d",&choice);

}while(choice!=5);

}

**Output:**

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).

//InfixToPostfix

#include <stdio.h>

#include <conio.h>

#include <ctype.h>

#include <string.h>

#define MAX 100 char st[MAX];int top = -1;void infixtopostfix(char source[], char target[]);

int getpriority(char);void push(char st[], char);

char pop(char st[]);int main(){char infix[100],

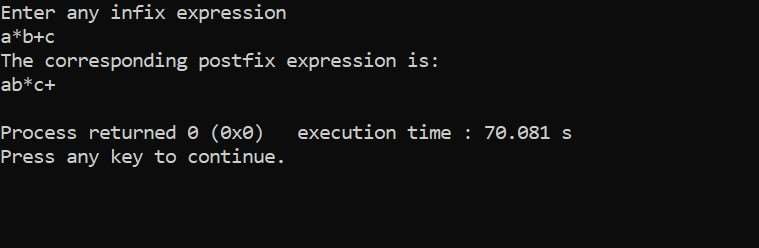
postfix[100];

printf("Enter any infix expression\n");

gets(infix);strcpy(postfix, "");

infixtopostfix(infix, postfix);printf("The corresponding postfix expression is:\n"); puts(postfix); return 0;}int getpriority(char op){if (op == '/' || op == '\*' || op == '%') return 1;else if (op == '+' || op == '-') return 0;}void push(char st[], char val){if (top == MAX - 1) printf("Stack overflow\n"); else{top++; st[top] = val;}}char pop(char st[]){char val = ' '; if (top == -1){printf("Stack Underflow\n"); }else{val = st[top]; top--;}return val;}void infixtopostfix(char source[], char target[]){int i = 0, j = 0; char temp; strcpy(target, "");while (source[i] != '\0'){if (source[i] == '('){push(st, source[i]); i++;}else if (source[i] == ')'){while ((top != -1) && (st[top] != '(')){target[j] = pop(st); j++; }if (top == -1){printf("\n Incorrect Expression"); exit(1);}temp = pop(st); i++;}else if (isdigit(source[i]) || isalpha(source[i])){target[j] = source[i]; j++;i++;}else if (source[i] == '+' || source[i] == '-' || source[i] == '\*' || source[i] == '/' || source[i] == '^'){while ((top != -1) && (st[top] != '(') && (getpriority(st[top]) > getpriority(source[i]))){target[j] = pop(st); j++;}push(st, source[i]); i++; }else{printf("\n Incorrect Element in Expression "); exit(1);}}while ((top != -1) && (st[top] != '(')){target[j] = pop(st); j++;}target[j] = '\0';}

Output:



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

int q[MAX]; int front = -1; int rear = -1; void insert(); int delete\_q(); void display(); int main()

{

while (1)

{

int choice,d;

printf("\n 1. insert \t 2.delete \t 3.display \t 4.exit\n"); scanf("%d", &choice);

switch (choice)

{

case 1: insert(); break; case 2:

d=delete\_q(); if (d!= -1)

printf("\n The number deleted is : %d", d); break;

case 3:

display(); break; case 4:

exit(0);

}

}

}

void insert()

{

if (rear == MAX - 1)

{

printf("Queue is Full\n"); return;

}

printf("Enter the element to be inserted\n"); int a;

scanf("%d", &a);

if ((front == -1) && (rear == -1))

{

front = rear = 0;

}

else

{

rear++;

}

q[rear] = a;

}

int delete\_q()

{

int val;

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

{

printf("Underflow\n"); return -1;

}

else{ val=q[front];

front++;

if(front>rear)

{

front=rear=-1;

}

return val;

}

}

void display()

{

printf("the elements are:\t"); for (int i = front; i <= rear; i++)

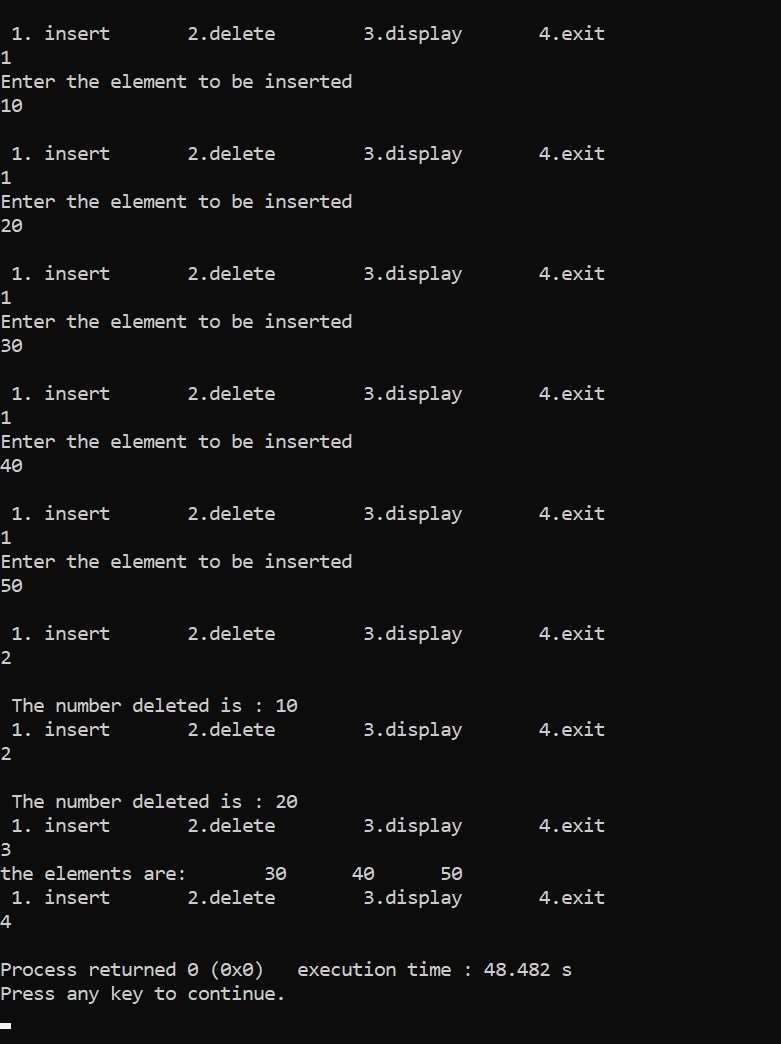
{

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

}

}

Output:



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

#include <stdio.h> #define MAX 5 int queue[MAX];

int front = -1, rear = -1; void insert();

int delete\_element(); int peek();

void display(); int main()

{

int option, val; do

{

printf("Enter : 1-Insert, 2-Delete, 3-Peek, 4-Display & 5-Exit : \n"); printf("Enter your option : \n");

scanf("%d", &option); switch (option)

{

case 1: insert(); break; case 2:

val = delete\_element(); if (val != -1)

printf("The number deleted is : %d \n", val); break;

case 3:

val = peek(); if (val != -1)

printf("\n The first value in queue is : %d \n", val); break;

case 4:

display(); break;

}

} while (option != 5); return 0;

}

void insert()

{

int num;

printf("Enter the number to be inserted in the queue : \n"); scanf("%d", &num);

if (front == 0 && rear == MAX - 1) printf(" OVERFLOW \n");

else if (front == -1 && rear == -1)

{

front = rear = 0;

queue[rear] = num;

}

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

{

rear = 0; queue[rear] = num;

}

else

{

rear++; queue[rear] = num;

}

}

int delete\_element()

{

int val;

if (front == -1 && rear == -1)

{

printf("UNDERFLOW \n"); return -1;

}

val = queue[front]; if (front == rear) front = rear = -1; else

{

if (front == MAX - 1) front = 0;

else front++;

if (front == -1 && rear == -1)

{

printf("QUEUE IS EMPTY \n"); return -1;

}

else

{

return queue[front];

}

}

void display()

{

int i;

//printf("\n");

if (front == -1 && rear == -1) printf("QUEUE IS EMPTY\n"); else

{

if (front < rear)

{

for (i = front; i <= rear; i++) printf("%d\t", queue[i]);

}

else

{

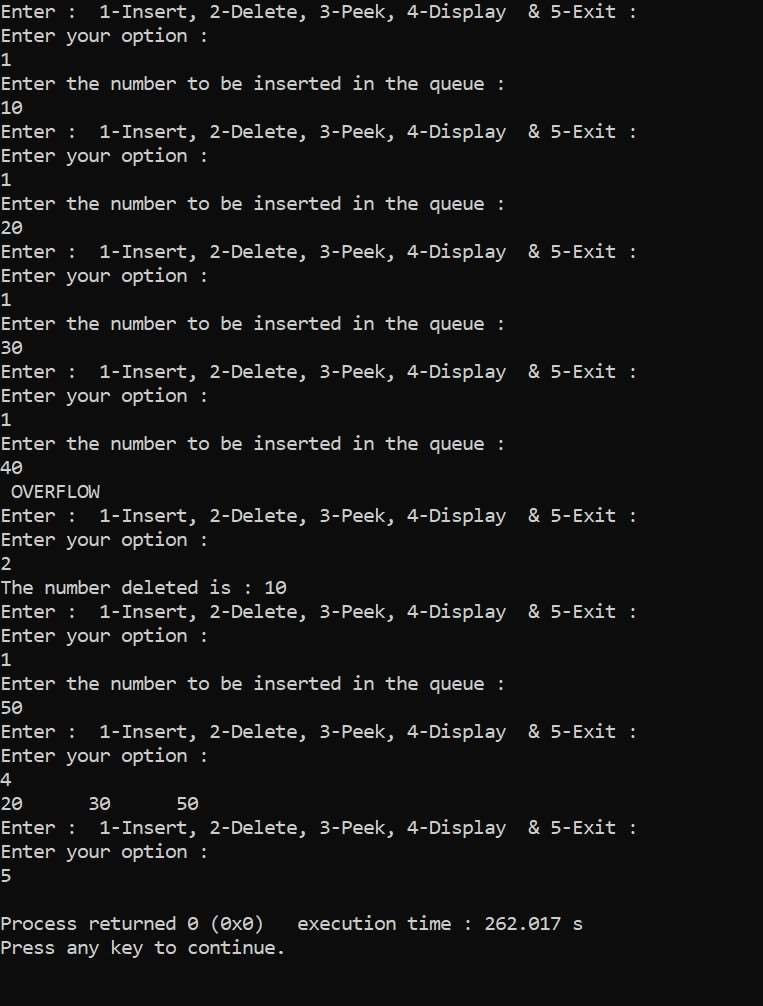
for (i = front; i < MAX; i++) printf("%d \t", queue[i]);

for (i = 0; i <= rear; i++) printf("%d \t ", queue[i]);

}

printf("\n");

}

Output:

4a) WAP to Implement Singly Linked List with following operations

1. Create a linked list.
2. 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;

};

void insertatbegin(struct node\*\* head,int item)

{

struct node\* newnode=(struct node\*)malloc(sizeof(struct node)); newnode->next=\*head;

newnode->data=item;

\*head=newnode;

}

void insertatend(struct node\*\* head,int item)

{

struct node\* newnode=(struct node\*)malloc(sizeof(struct node)); struct node\* temp =\*head;

newnode->next=NULL; newnode->data=item; if(\*head==NULL)

{

\*head=newnode; return;

}

while(temp->next!=NULL)

{

temp=temp->next;

}

temp->next=newnode;

}

void insertatspecific(struct node\*\* head,int item,int loc)

{

if(loc<=0)

{

printf("invalid position\n"); return;

}

if(loc==1 || \*head==NULL)

{

insertatbegin(head,item); return;

}

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

struct node\* temp =\*head; int count=1;

while(count<loc-1 && temp->next!=NULL)

{

temp=temp->next; count++;

}

newnode->next=temp->next; temp->next=newnode;

}

void display(struct node\* head)

{

struct node\* temp=head; if(temp==NULL)

{

printf("linked list is empty\n"); return;

}

while(temp!=NULL)

{

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

}

printf("NULL\n");

}

int main()

{

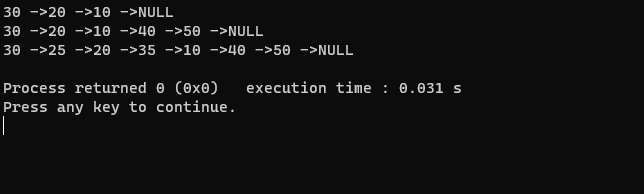
struct node\* head=NULL;

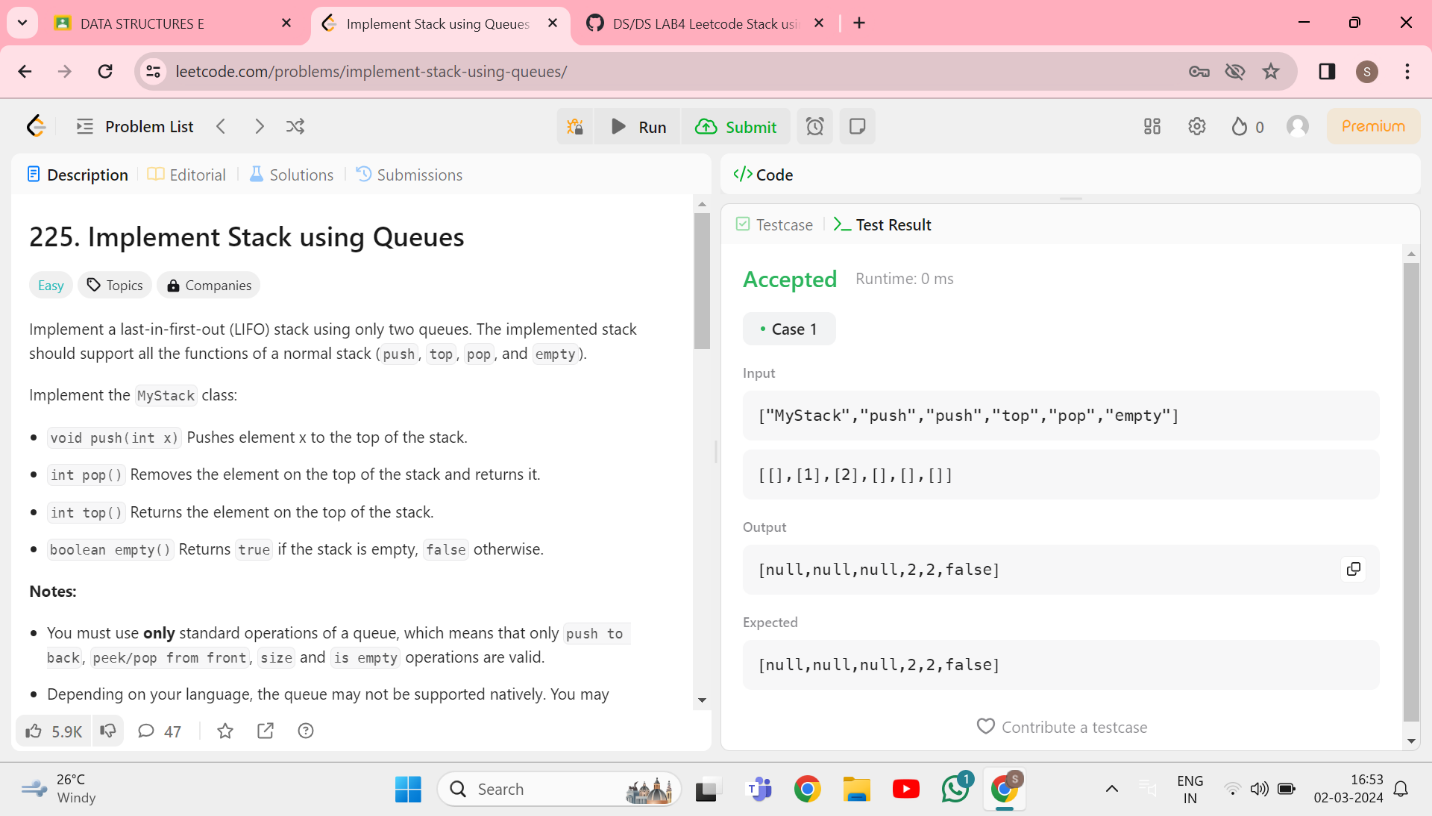
insertatbegin(&head,10); insertatbegin(&head,20); insertatbegin(&head,30); insertatend(&head,40); insertatend(&head,50); insertatspecific(&head,25,2); insertatspecific(&head,35,4); display(head);

return 0;

}

Output:



4b) Program - Leetcode platform

#include <stdbool.h>

#include <stdlib.h>

typedef struct {

int\* data;

int front;

int rear;

int size;

} Queue;

typedef struct {

Queue\* q1;

Queue\* q2;

} MyStack;

Queue\* createQueue(int size) {

Queue\* queue = (Queue\*)malloc(sizeof(Queue));

queue->data = (int\*)malloc(size \* sizeof(int));

queue->front = queue->rear = -1;

queue->size = size;

return queue;

}

void enqueue(Queue\* queue, int value) {

if (queue->rear == -1) {

queue->front = queue->rear = 0;

} else {

queue->rear = (queue->rear + 1) % queue->size;

}

queue->data[queue->rear] = value;

}

int dequeue(Queue\* queue) {

int value = queue->data[queue->front];

if (queue->front == queue->rear) {

queue->front = queue->rear = -1;

} else {

queue->front = (queue->front + 1) % queue->size;

}

return value;

}

bool isEmpty(Queue\* queue) {

return queue->front == -1;

}

MyStack\* myStackCreate() {

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

stack->q1 = createQueue(1000); // Adjust the size as needed

stack->q2 = createQueue(1000);

return stack;

}

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

enqueue(obj->q1, x);

}

int myStackPop(MyStack\* obj) {

if (isEmpty(obj->q1)) {

return -1; // Stack is empty

}

while (obj->q1->front != obj->q1->rear) {

enqueue(obj->q2, dequeue(obj->q1));

}

int poppedValue = dequeue(obj->q1);

// Swap q1 and q2

Queue\* temp = obj->q1;

obj->q1 = obj->q2;

obj->q2 = temp;

return poppedValue;

}

int myStackTop(MyStack\* obj) {

if (isEmpty(obj->q1)) {

return -1; // Stack is empty

}

while (obj->q1->front != obj->q1->rear) {

enqueue(obj->q2, dequeue(obj->q1));

}

int topValue = dequeue(obj->q1);

enqueue(obj->q2, topValue);

// Swap q1 and q2

Queue\* temp = obj->q1;

obj->q1 = obj->q2;

obj->q2 = temp;

return topValue;

}

bool myStackEmpty(MyStack\* obj) {

return isEmpty(obj->q1);

}

void myStackFree(MyStack\* obj) {

free(obj->q1->data);

free(obj->q1);

free(obj->q2->data);

free(obj->q2);

free(obj);

}

5a) WAP to Implement Singly Linked List with following operations

1. Create a linked list.
2. Deletion of first element, specified element and last element in the list.Display the contents of the linked list.

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

int data;

struct Node\* next;

};

void insertAtBeginning(struct Node\*\* head, int value) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node)); newNode->data = value;

newNode->next = \*head;

\*head = newNode;

}

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

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

struct Node\* temp = \*head;

newNode->data = value; newNode->next = NULL; if (\*head == NULL) {

\*head = newNode; return;

}

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

}

temp->next = newNode;

}

void insertAtPosition(struct Node\*\* head, int value, int position) { if (position <= 0) {

printf("Invalid position\n"); return;

}

if (position == 1 || \*head == NULL) { insertAtBeginning(head, value);

return;

}

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node)); newNode->data = value;

struct Node\* temp = \*head; int count = 1;

while (count < position - 1 && temp->next != NULL) {

temp = temp->next;

count++;

}

if (count < position - 1) { printf("Invalid position\n"); return;

}

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

}

void deleteAtBegining(struct Node\*\* head){ if (\*head == NULL) {

printf("The linkedlist is already empty\n"); return;

}

else{

struct Node\* first = \*head;

\*head = (\*head)->next; free(first);

}

}

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

if(\*head==NULL) {

printf("The linkedlist is already empty\n"); return;

}

else{

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

}

struct Node\* lastNode = temp->next; temp->next=NULL;

free(lastNode);

}

}

void deleteAtIndex(struct Node \*\*head, int pos) { if(\*head == NULL){

printf("The Linked List is Empty \n");

}

else{

struct Node\* temp = \*head; pos--;

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

}

if(temp==NULL){ printf("pos not exist\n");

}

else{

struct Node\* nxt = temp->next->next; struct Node\* del = temp->next;

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

}

}

}

void displayLinkedList(struct Node\* head) { struct Node\* temp = head;

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

}

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

}

printf("NULL\n");

}

int main() {

struct Node\* head = NULL;

insertAtBeginning(&head, 5);

insertAtBeginning(&head, 3);

insertAtBeginning(&head, 1);

printf("Linked list after insertion at the beginning: "); displayLinkedList(head);

insertAtEnd(&head, 6);

insertAtEnd(&head, 7);

printf("Linked list after insertion at the end: "); displayLinkedList(head); insertAtPosition(&head, 2, 2);

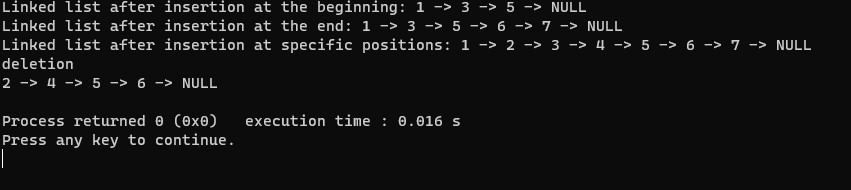
insertAtPosition(&head, 4, 4);

printf("Linked list after insertion at specific positions: "); displayLinkedList(head);

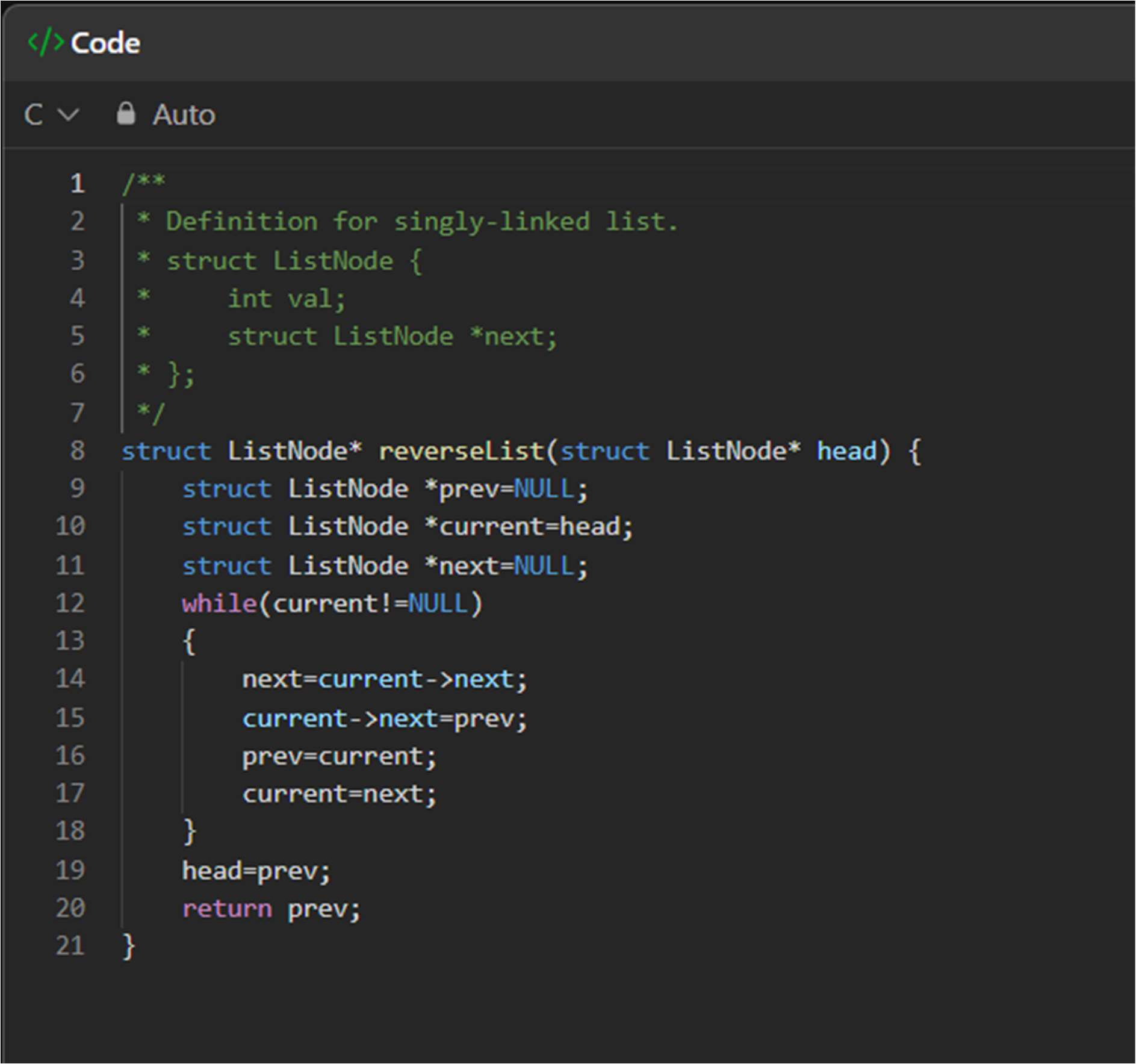
printf("deletion\n"); deleteAtBegining(&head); deleteAtIndex(&head,1); deleteAtEnd(&head); displayLinkedList(head); return 0;

}

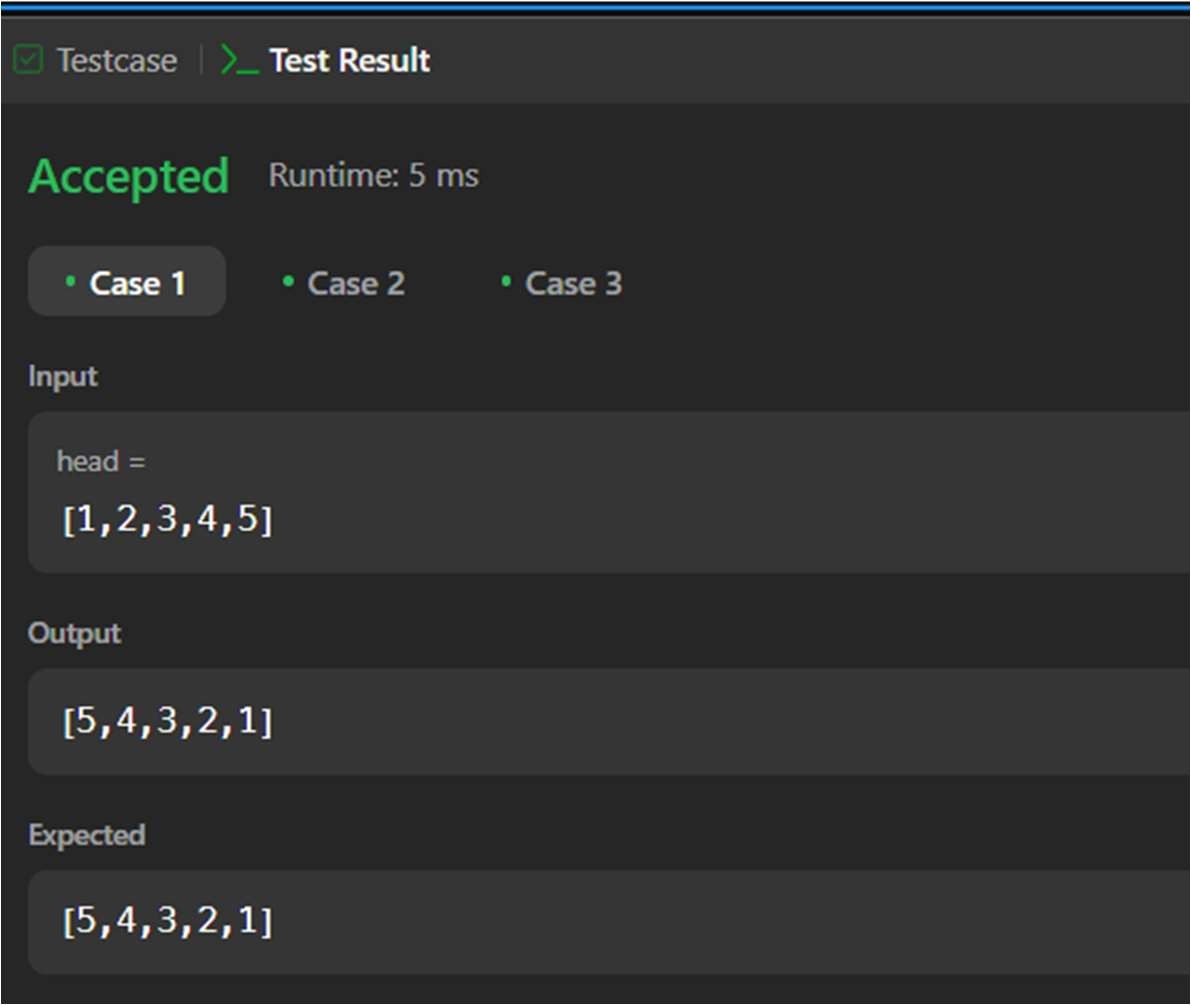
OUTPUT:

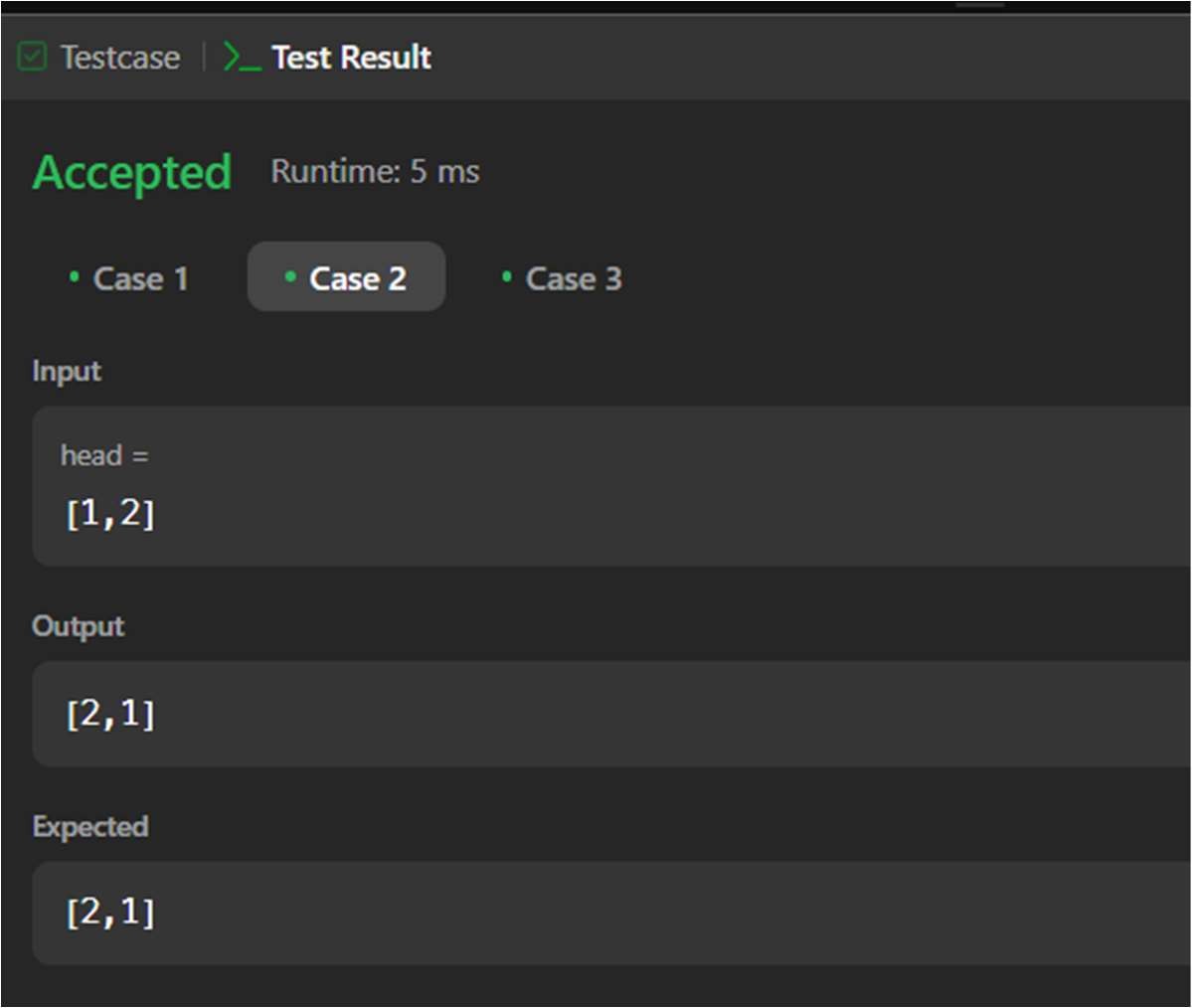


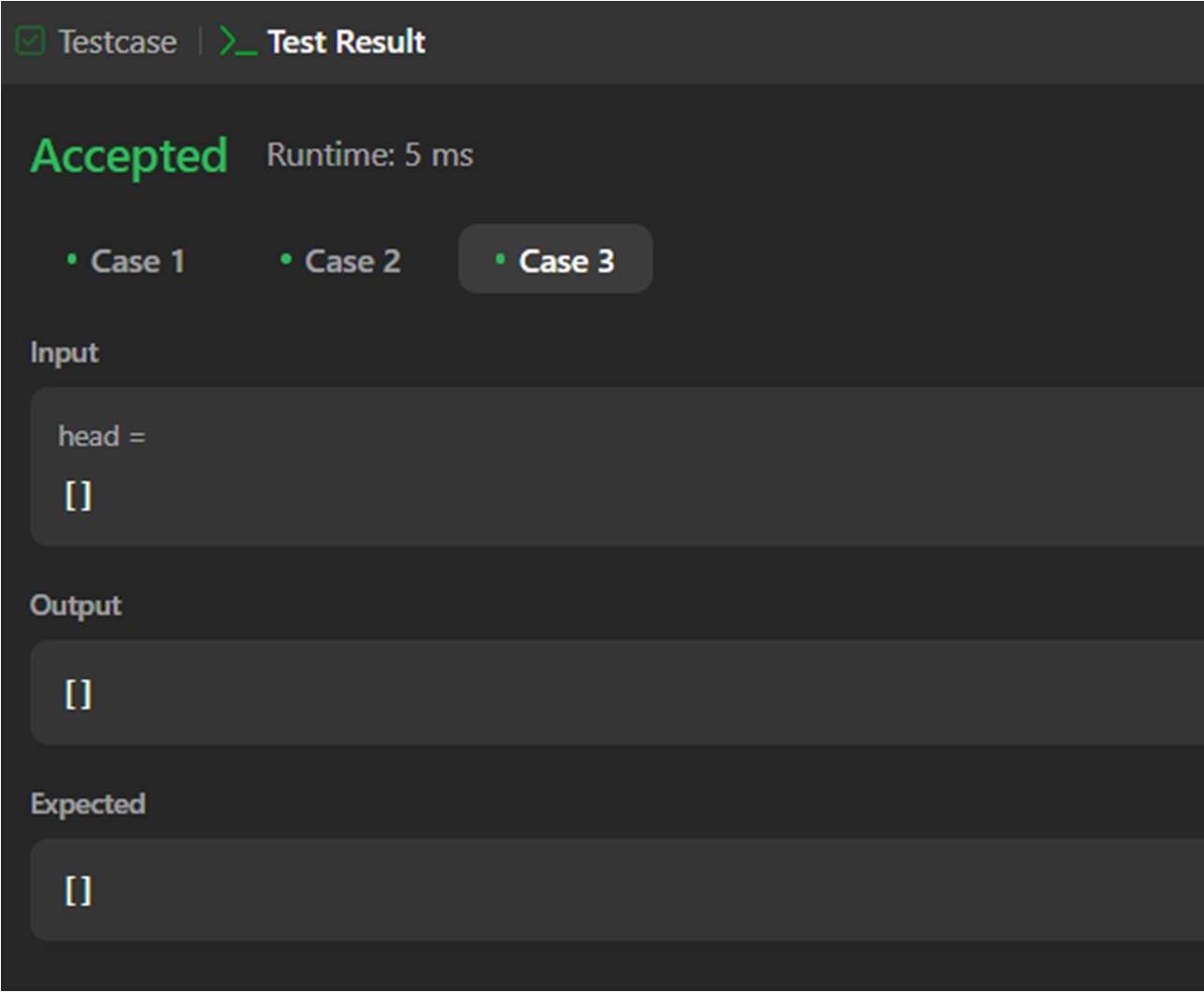
5b) Program - Leetcode platform



OUTPUT:







6a)WAP to Implement Singly Linked 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;

};

void insertatbegin(struct node\*\* head,int value)

{ struct node\* new\_node=(struct node\*)malloc(sizeof(struct node)); new\_node->data=value;

new\_node->next=\*head;

\*head=new\_node;

}

void concat(struct node \*head1,struct node \*head2)

{

if (head1->next == NULL) head1->next = head2;

else

concat(head1->next,head2);

}

void sortlist(struct node\*\* head1)

{

struct node \*temp,\*i; for(temp=\*head1;temp!=NULL;temp=temp->next)

{

for(i=temp->next;i!=NULL;i=i->next)

{

if(i->data < temp->data)

{ int tem=i->data;

i->data=temp->data; temp->data=tem;

}

}

}

}

void reverse(struct node\*\* head1)

{

struct node \*prev=NULL; struct node \*current=\*head1; struct node\* next=NULL; while(current!=NULL)

{

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

}

\*head1=prev;

}

void printlist(struct node\* node)

{

struct node\* temp=node; while(temp!=NULL)

{

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

}

printf("NULL\n");

}

int main()

{

struct node \*head1=NULL; insertatbegin(&head1,10); insertatbegin(&head1,15); insertatbegin(&head1,40); insertatbegin(&head1,50); printf("List 1:"); printlist(head1);

struct node \*head2=NULL; insertatbegin(&head2,65); insertatbegin(&head2,75); insertatbegin(&head2,60); printf("List 2:"); printlist(head2); concat(head1,head2);

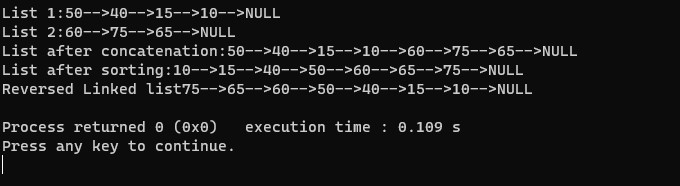
printf("List after concatenation:");

printlist(head1); sortlist(&head1); printf("List after sorting:"); printlist(head1); reverse(&head1);

printf("Reversed Linked list"); printlist(head1);

}

OUTPUT:



6b) WAP to Implement Singly Linked List to simulate Stack & Queue Operations.

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

int data;

struct Node\* next;

};

void insertAtBeginning(struct Node\*\* head, int value) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node)); newNode->data = value;

newNode->next = \*head;

\*head = newNode;

}

void deleteAtBeginning(struct Node\*\* head) { if (\*head == NULL) {

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

}

struct Node\* temp = \*head;

\*head = (\*head)->next; free(temp);

}

void display(struct Node\* head)

{

struct Node\* temp = head; if (temp == NULL) {

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

}

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

}

printf("NULL\n");

}

int main()

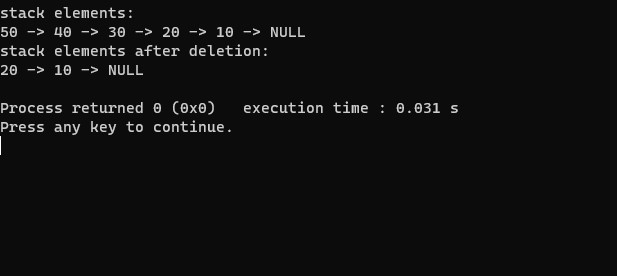
{

struct Node\* head=NULL; insertAtBeginning(&head,10); insertAtBeginning(&head,20); insertAtBeginning(&head,30); insertAtBeginning(&head,40); insertAtBeginning(&head,50); printf("stack elements:\n"); display(head); deleteAtBeginning(&head); deleteAtBeginning(&head); deleteAtBeginning(&head);

printf("stack elements after deletion:\n"); display(head);

return 0;

}

OUTPUT:

7a) WAP to Implement doubly link list with primitive operations

1. Create a doubly linked list.
2. Insert a new node to the left of the node.
3. Delete the node based on a specific value 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 data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node)); if (newNode == NULL) {

printf("Memory allocation failed\n"); exit(EXIT\_FAILURE);

}

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

}

void insertAtBeginning(struct Node\*\* head, int data) { struct Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;

} else {

newNode->next = \*head; (\*head)->prev = newNode;

\*head = newNode;

}

}

void insertBeforeNode(struct Node\*\* head, int key, int data) { if (\*head == NULL) {

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

}

struct Node\* newNode = createNode(data); struct Node\* current = \*head;

while (current) {

if (current->data == key) { if (current->prev) {

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

} else {

\*head = newNode;

}

newNode->next = current; current->prev = newNode; return;

}

current = current->next;

}

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

}

void deleteNode(struct Node\*\* head, int pos) {

if (\*head == NULL) { printf("List is empty\n"); return;

}

struct Node\* current = \*head; int count = 1;

while (current && count < pos) { current = current->next; count++;

}

if (current == NULL) {

printf("Position %d is beyond the length of the list\n", pos); return;

}

if (current->prev) {

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

} else {

\*head = current->next;

}

if (current->next) {

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

}

free(current);

printf("Node at position %d deleted\n", pos);

}

void displayList(struct Node\* head) { if (head == NULL) {

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

}

struct Node\* current = head; while (current) {

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

}

printf("\n");

}

void freeList(struct Node\* head) { struct Node\* current = head; struct Node\* nextNode;

while (current) { nextNode = current->next; free(current);

current = nextNode;

}

}

int main() {

struct Node\* head = NULL; int ch, newData, pos, key; while (1) { printf("\nMenu\n");

printf("1. Insert at the beginning\n"); printf("2. Insert before a node\n"); printf("3. Delete a node\n"); printf("4. Display list\n");

printf("5. Free doubly linked list and exit\n"); printf("Enter your choice: ");

scanf("%d", &ch); switch (ch) {

case 1:

printf("Enter data to insert at the beginning: "); scanf("%d", &newData); insertAtBeginning(&head, newData);

break; case 2:

printf("Enter the value before which you want to insert: ");

scanf("%d", &key); printf("Enter data to insert: "); scanf("%d", &newData);

insertBeforeNode(&head, key, newData); break;

case 3:

printf("Enter the position you wish to delete: "); scanf("%d", &key);

deleteNode(&head, key); break;

case 4:

printf("Doubly linked list: "); displayList(head);

break; case 5:

freeList(head);

printf("Exiting the program\n"); return 0;

default:

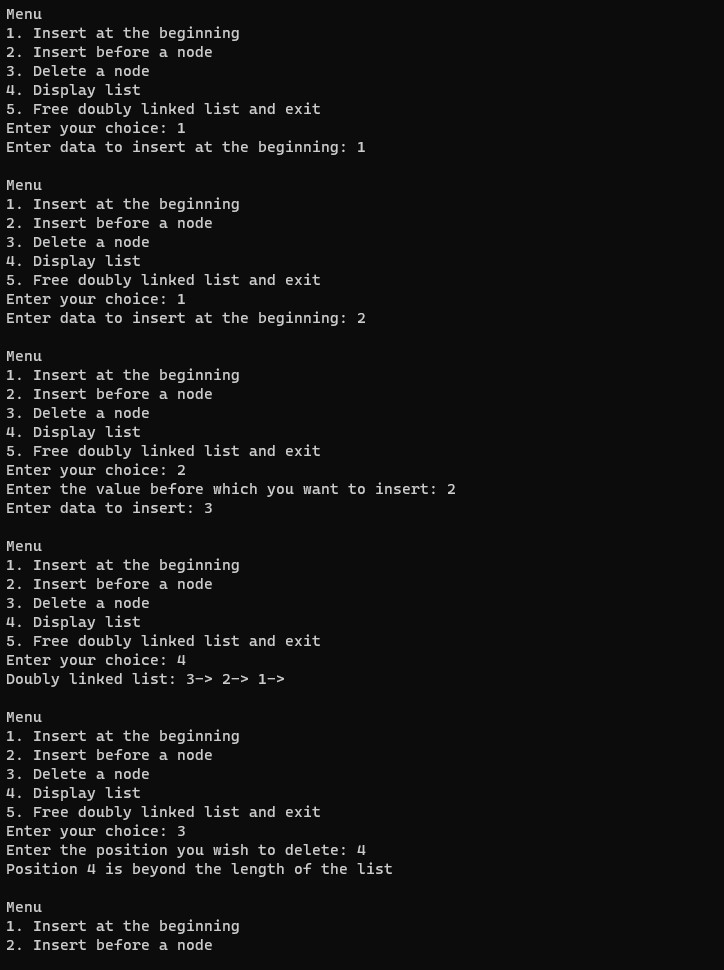
printf("Invalid choice\n");

}

}

return 0;

}

OUTPUT:

7b) Program – hackerrank platform

Q) Given pointers to the heads of two sorted linked lists, merge them into a single, sorted linked list. Either head pointer may be null meaning that the corresponding list is empty.

A) SinglyLinkedListNode\* mergeLists(SinglyLinkedListNode\* head1, SinglyLinkedListNode\* head2) {

     if (head1 == NULL) {

        return head2;

    } else if (head2 == NULL) {

        return head1;

    } else if (head1->data <= head2->data) {

        head1->next = mergeLists(head1->next, head2);

        return head1;

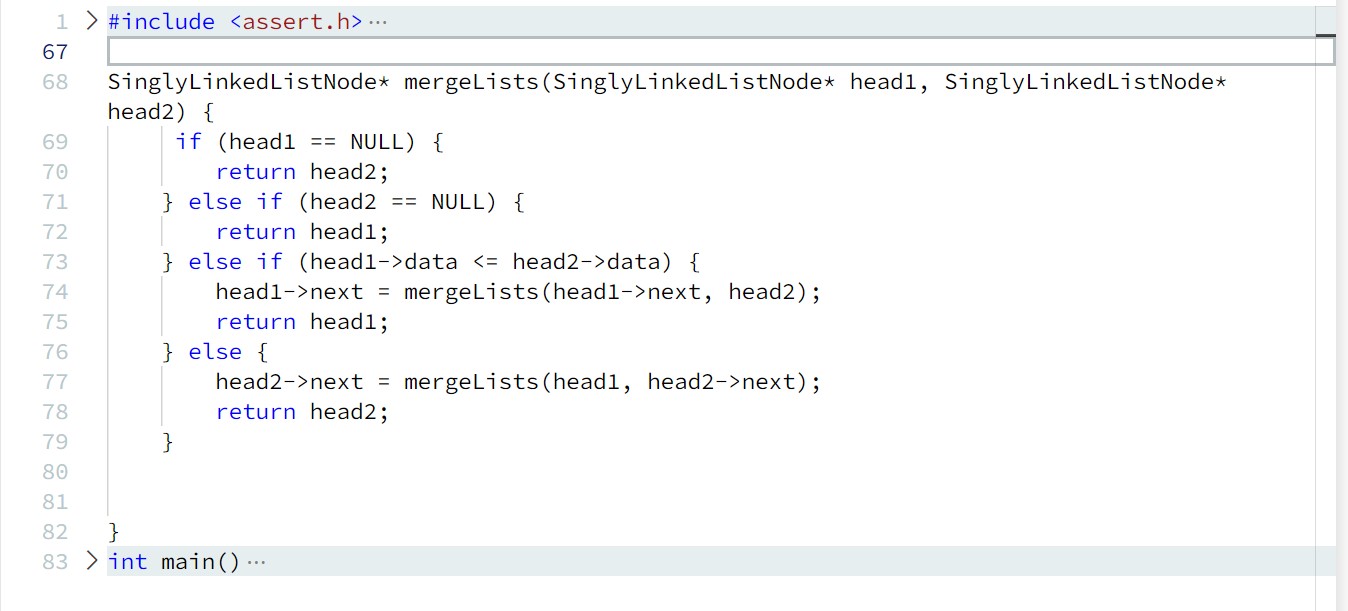
    } else {

        head2->next = mergeLists(head1, head2->next);

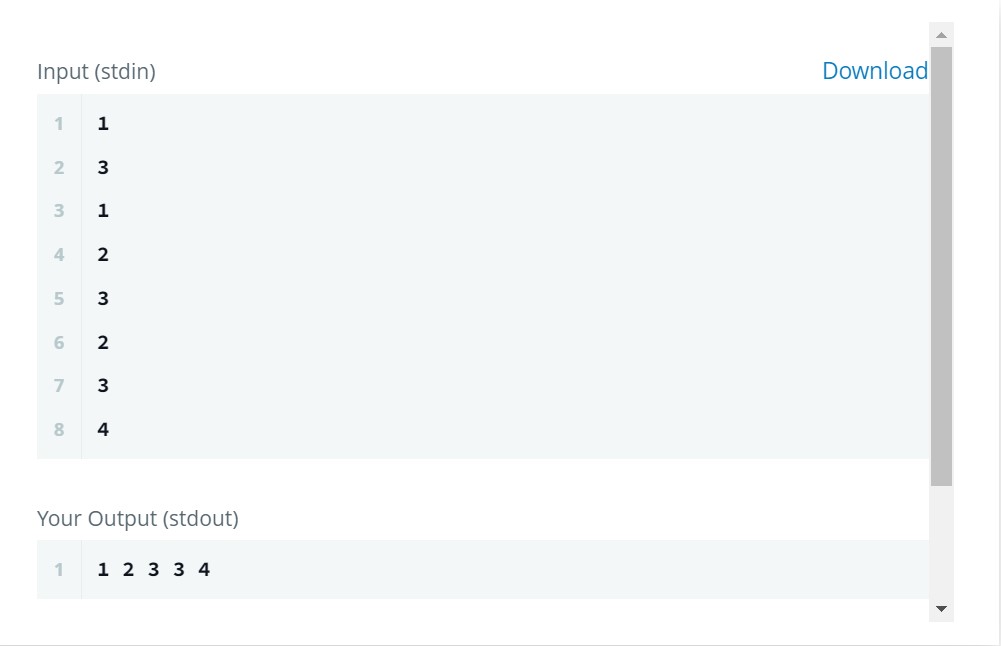
        return head2;

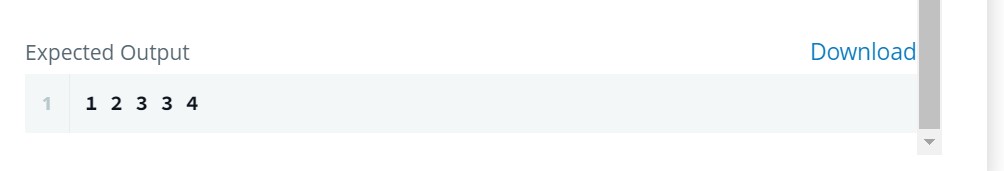
    }

}



OUTPUT:





8a) Write a program

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

To display the elements in the tree.

#include <stdio.h>

#include <stdlib.h>

typedef struct node

{

int data;

struct node \*leftnode;

struct node \*rightnode;

} node;

node \*create(int data)

{

node \*a;

a = malloc(sizeof(node));

a->data = data;

a->leftnode = NULL;

a->rightnode = NULL;

return a;

}

void preordertraversal(node \*root)

{

if (root != NULL)

{

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

preordertraversal(root->leftnode);

preordertraversal(root->rightnode);

}

}

void postordertraversal(node \*root)

{

if (root != NULL)

{

postordertraversal(root->leftnode);

postordertraversal(root->rightnode);

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

}

}

void inordertraversal(node \*root)

{

if (root != NULL)

{

inordertraversal(root->leftnode);

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

inordertraversal(root->rightnode);

}

}

int isBST(node \*root)

{

static node \*prev = NULL;

if (root != NULL)

{

if (!isBST(root->leftnode))

{

return 0;

}

else if (prev != NULL && root->data <= prev->data)

{

return 0;

}

prev = root;

return isBST(root->rightnode);

}

else

return 1;

}

node \*search\_iterative(node \*root, int key)

{

if (root == NULL)

{

return NULL;

}

while (root != NULL)

{

if (key == root->data)

{

return root;

}

else if (key < root->data)

{

root = root->leftnode;

}

else if (key > root->data)

{

root = root->rightnode;

}

}

return NULL; // Return NULL if the key is not found

}

void insert(node \*\*root, int key)

{

node \*news, \*prev = NULL;

news = create(key);

if (\*root == NULL)

{

\*root = news;

return;

}

node \*temp = \*root; // Use a temporary variable for traversal

while (temp != NULL)

{

prev = temp;

if (key == temp->data)

{

printf("cannot insert");

free(news); // Free the allocated memory before returning

return;

}

else if (key < temp->data)

{

temp = temp->leftnode;

}

else if (key > temp->data)

{

temp = temp->rightnode;

}

}

if (key < prev->data)

{

prev->leftnode = news;

}

else if (key > prev->data)

{

prev->rightnode = news;

}

}

node \*inorderpredecessor(node \*root)

{

root = root->leftnode;

while (root->rightnode != NULL)

{

root = root->rightnode;

}

return root;

}

node \*deletenode(node \*root, int value)

{

node \*ipre;

if (root == NULL)

{

return NULL;

}

if (value < root->data)

{

root->leftnode = deletenode(root->leftnode, value);

}

else if (value > root->data)

{

root->rightnode = deletenode(root->rightnode, value);

}

else

{

if (root->leftnode == NULL)

{

node \*temp = root->rightnode;

free(root);

return temp;

}

else if (root->rightnode == NULL)

{

node \*temp = root->leftnode;

free(root);

return temp;

}

ipre = inorderpredecessor(root);

root->data = ipre->data;

root->leftnode = deletenode(root->leftnode, ipre->data);

}

return root;

}

int main()

{

node \*root = create(50);

node \*leafl = create(45);

node \*leafr = create(60);

node \*leafll = create(35);

node \*leafrr = create(65);

node \*leaflr = create(47);

node \*leafrl = create(55);

root->leftnode = leafl;

root->rightnode = leafr;

leafl->leftnode = leafll;

leafr->rightnode = leafrr;

leafl->rightnode = leaflr;

leafr->leftnode = leafrl;

insert(&root, 70);

printf("Preorder Traversal: ");

preordertraversal(root);

printf("\n");

printf("Postorder Traversal: ");

postordertraversal(root);

printf("\n");

printf("Inorder Traversal: ");

inordertraversal(root);

printf("\n");

printf("Is BST: %d\n", isBST(root));

int keyToSearch = 80;

node \*n = search\_iterative(root, keyToSearch);

if (n != NULL)

{

printf("Element %d found\n", n->data);

}

else

{

printf("Element not found\n");

}

printf("Deleting node with value 50\n");

root = deletenode(root, 50);

printf("After deletion\n");

printf("Preorder Traversal: ");

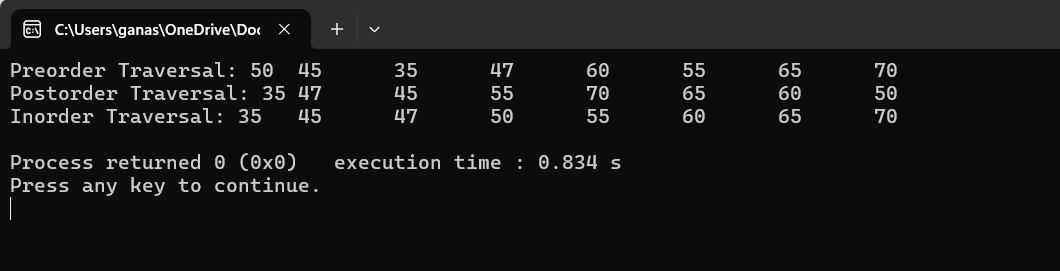
preordertraversal(root);

printf("\n");

return 0;

}

Output:



8b) Program - Leetcode platform

Q) Merge two Binary Trees

struct TreeNode\* mergeTrees(struct TreeNode\* root1, struct TreeNode\* root2) {

     if(root1 == NULL)

        {

            return root2;

        }

        if(root2 == NULL)

        {

            return root1;

        }

        root1->val += root2->val;

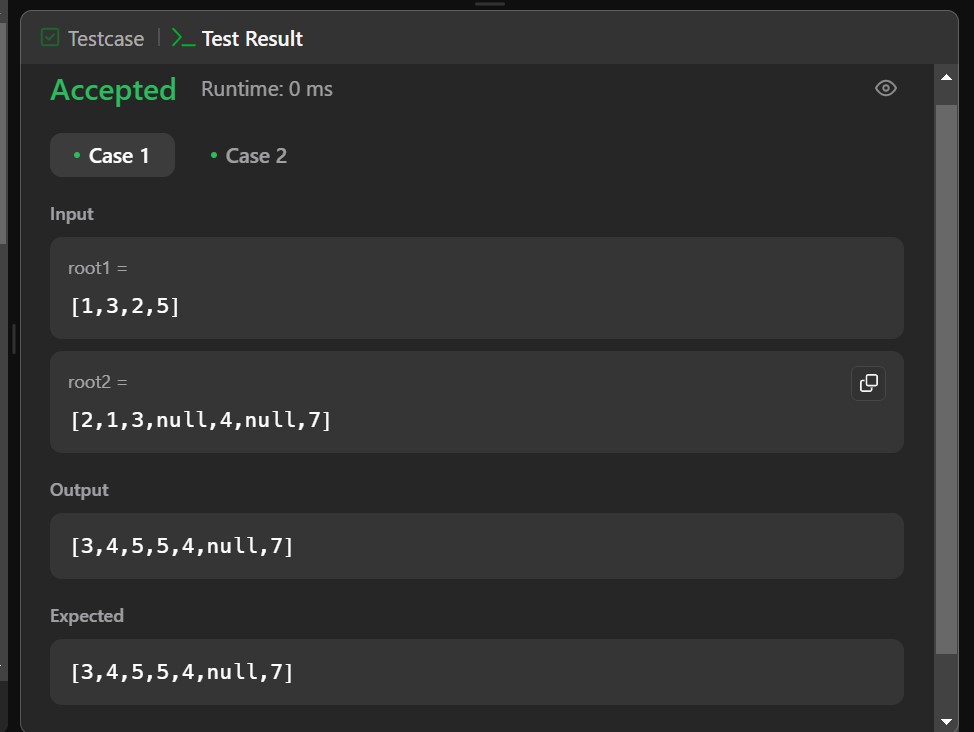
        root1->left = mergeTrees(root1->left,root2->left);

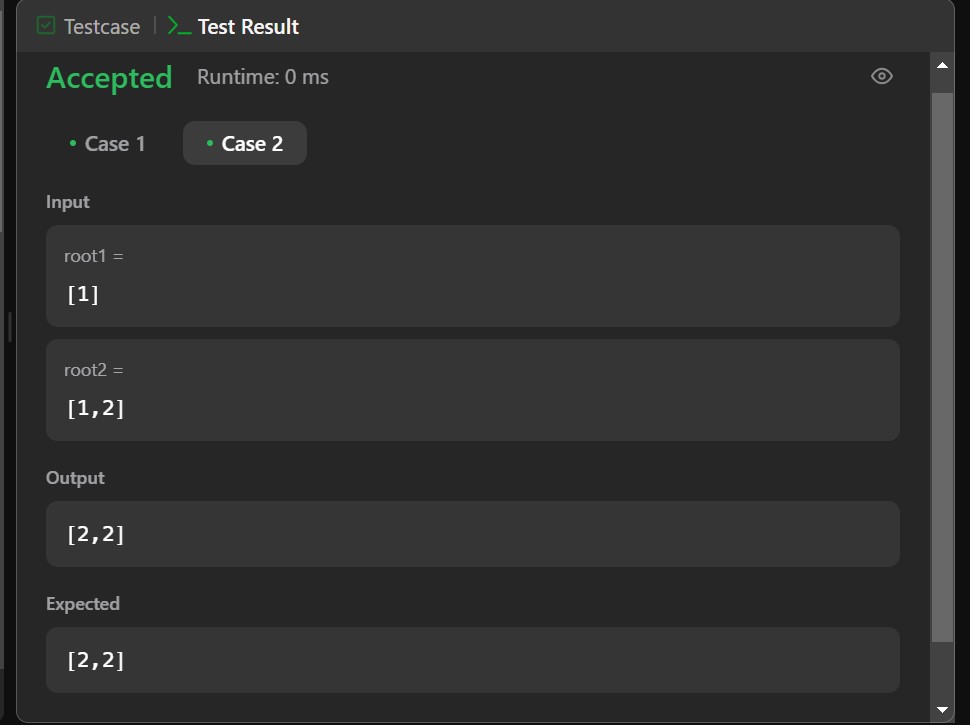
        root1->right = mergeTrees(root1->right,root2->right);

        return root1;

}

OUTPUT:





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

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

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define MAX\_NODES 100

struct Queue {

int items[MAX\_NODES];

int front;

int rear;

};

struct Graph {

int vertices;

int\*\* adjMatrix;

};

struct Queue\* createQueue() {

struct Queue\* queue = (struct Queue\*)malloc(sizeof(struct Queue));

queue->front = -1;

queue->rear = -1;

return queue;

}

void enqueue(struct Queue\* queue, int value) {

if (queue->rear == MAX\_NODES - 1) {

printf("Queue is full\n");

} else {

if (queue->front == -1) {

queue->front = 0;

}

queue->rear++;

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

}

}

int dequeue(struct Queue\* queue) {

int item;

if (queue->front == -1) {

printf("Queue is empty\n");

item = -1;

} else {

item = queue->items[queue->front];

queue->front++;

if (queue->front > queue->rear) {

queue->front = queue->rear = -1;

}

}

return item;

}

bool isEmpty(struct Queue\* queue) {

return queue->front == -1;

}

struct Graph\* createGraph(int vertices) {

struct Graph\* graph = (struct Graph\*)malloc(sizeof(struct Graph));

graph->vertices = vertices;

graph->adjMatrix = (int\*\*)malloc(vertices \* sizeof(int\*));

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

graph->adjMatrix[i] = (int\*)malloc(vertices \* sizeof(int));

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

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

}

}

return graph;

}

void addEdge(struct Graph\* graph, int src, int dest) {

graph->adjMatrix[src][dest] = 1;

graph->adjMatrix[dest][src] = 1;

}

void BFS(struct Graph\* graph, int startNode) {

struct Queue\* queue = createQueue();

bool visited[MAX\_NODES] = {false};

printf("BFS traversal starting from node %d: ", startNode);

visited[startNode] = true;

printf("%d ", startNode);

enqueue(queue, startNode);

while (!isEmpty(queue)) {

int currentNode = dequeue(queue);

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

if (graph->adjMatrix[currentNode][i] == 1 && !visited[i]) {

printf("%d ", i);

visited[i] = true;

enqueue(queue, i);

}

}

}

printf("\n");

}

bool isCyclicUtil(struct Graph\* graph, int v, bool visited[], int parent);

bool isCyclic(struct Graph\* graph) {

bool\* visited = (bool\*)malloc(graph->vertices \* sizeof(bool));

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

visited[i] = false;

}

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

if (!visited[i]) {

if (isCyclicUtil(graph, i, visited, -1)) {

free(visited);

return true;

}

}

}

free(visited);

return false;

}

bool isCyclicUtil(struct Graph\* graph, int v, bool visited[], int parent) {

visited[v] = true;

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

if (graph->adjMatrix[v][i] == 1) {

if (!visited[i]) {

if (isCyclicUtil(graph, i, visited, v)) {

return true;

}

} else if (i != parent) {

return true;

}

}

}

return false;

}

int main() {

struct Graph\* graph = createGraph(4);

addEdge(graph, 0, 1);

addEdge(graph, 0, 2);

addEdge(graph, 1, 2);

addEdge(graph, 2, 0);

addEdge(graph, 2, 3);

addEdge(graph, 3, 3);

int startNode = 2;

BFS(graph, startNode);

if (isCyclic(graph)) {

printf("The graph contains a cycle.\n");

} else {

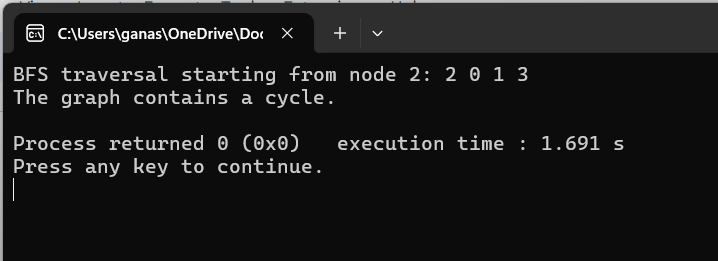
printf("The graph does not contain a cycle.\n");

}

return 0;

}

Output:



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 -> 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>

#define M 100 // number of memory locations (m)

#define N 1000 // number of employee records (N)

#define K 10000 // number of keys (K)

typedef struct {

int id;

char name[50];

float salary;

} Employee;

typedef struct Node {

int key;

Employee emp;

struct Node \*next;

} Node;

Node \*HT[M]; // Hash Table (HT) of m memory locations

// Hash function to map key K to address space L

int hash(int key) {

return key % M;

}

// Function to insert employee record into Hash Table

void insert(Employee emp) {

int key = emp.id;

int index = hash(key);

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

newNode->key = key;

newNode->emp = emp;

newNode->next = NULL;

if (HT[index] == NULL) {

HT[index] = newNode;

} else {

Node \*temp = HT[index];

while (temp->next != NULL) {

if (temp->key == key) {

printf("Collision detected! Updating employee record...\n");

temp->emp = emp;

free(newNode);

return;

}

temp = temp->next;

}

if (temp->key == key) {

printf("Collision detected! Updating employee record...\n");

temp->emp = emp;

free(newNode);

return;

}

temp->next = newNode;

}

}

// Function to search employee record using key K

void search(int key) {

int index = hash(key);

Node \*temp = HT[index];

while (temp != NULL) {

if (temp->key == key) {

printf("Employee Record Found:\n");

printf("ID: %d\n", temp->emp.id);

printf("Name: %s\n", temp->emp.name);

printf("Salary: %.2f\n", temp->emp.salary);

return;

}

temp = temp->next;

}

printf("Employee Record Not Found!\n");

}

int main() {

// Insert employee records into Hash Table

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

Employee emp;

emp.id = i + 1;

sprintf(emp.name, "Employee %d", i + 1);

emp.salary = (float)(i + 1) \* 1000;

insert(emp);

}

// Search employee record using key K

int key = 500;

search(key);

return 0;

}

Output:

