ASSIGNMENT- 02

Formative Assessment-LAB

Section-2

#include <stdio.h>

#define SIZE 5 // Maximum size of the queue

int queue[SIZE];

int front = -1, rear = -1;

int isFull() {

return (front == 0 && rear == SIZE - 1) || (front == rear + 1);

}

int isEmpty() {

return (front == -1);

}

void enqueue(int callID) {

if (isFull()) {

printf("Queue is full. Please wait.\n");

return;

}

if (front == -1)

front = 0;

rear = (rear + 1) % SIZE;

queue[rear] = callID;

printf("Call added: %d\n", callID);

}

void dequeue() {

if (isEmpty()) {

printf("Queue is empty. No calls to remove.\n");

return;

}

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

if (front == rear) {

front = rear = -1;

} else {

front = (front + 1) % SIZE;

}

}

void displayQueue() {

if (isEmpty()) {

printf("No customers in queue.\n");

return;

}

printf("Customers in queue: ");

int i = front;

while (1) {

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

if (i == rear)

break;

i = (i + 1) % SIZE;

}

printf("\n");

}

int main() {

enqueue(101);

enqueue(102);

enqueue(103);

enqueue(104);

enqueue(105);

displayQueue();

enqueue(106);

dequeue();

dequeue();

displayQueue();

enqueue(106);

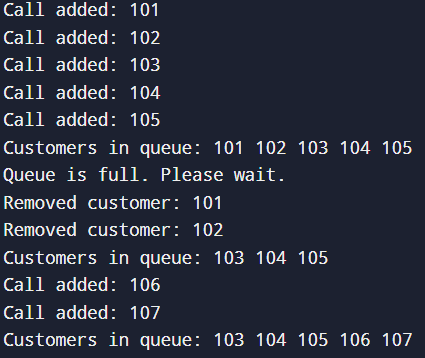
enqueue(107);

displayQueue();

return 0;

}

OUTPUT-



Section-3

#include <stdio.h>

#define MAX 10

int undoStack[MAX];

int redoStack[MAX];

int topUndo = -1;

int topRedo = -1;

void performOperation(int op) {

if (topUndo < MAX - 1) {

undoStack[++topUndo] = op;

topRedo = -1;

printf("Performed operation: \"op%d\"\n", op);

}

}

void undoOperation() {

if (topUndo >= 0) {

int op = undoStack[topUndo--];

redoStack[++topRedo] = op;

if (topUndo >= 0) {

printf("Undone. Next Operation that can be undone is = \"op%d\"\n", undoStack[topUndo]);

} else {

printf("Undone. No more operations to undo.\n");

}

} else {

printf("Nothing to undo.\n");

}

}

void redoOperation() {

if (topRedo >= 0) {

int op = redoStack[topRedo--];

undoStack[++topUndo] = op;

if (topRedo >= 0) {

printf("Redo completed. Next Operation that can be redone is = \"op%d\"\n", redoStack[topRedo]);

} else {

printf("Redo completed. No more operations to redo.\n");

}

} else {

printf("Nothing to redo.\n");

}

}

int main() {

performOperation(1);

performOperation(2);

performOperation(3);

performOperation(4);

undoOperation();

undoOperation();

redoOperation();

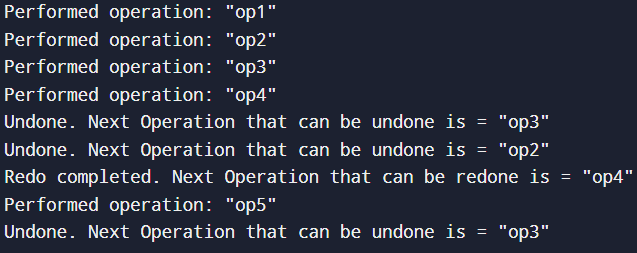
performOperation(5);

undoOperation();

return 0;

}

OUTPUT-



Formative Assessment- Theory

QUESTION-01

#include <stdio.h>

#include <string.h>

#define DAYS 30

typedef struct {

char name[50];

int steps[DAYS];

} Member;

void analyzeSteps(Member members[], int n) {

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

int count = 0, maxSteps = members[i].steps[0];

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

if (members[i].steps[j] > 10000)

count++;

if (members[i].steps[j] > maxSteps)

maxSteps = members[i].steps[j];

}

printf("Member: %s\n", members[i].name);

printf("Days exceeding 10,000 steps: %d\n", count);

printf("Maximum steps in month: %d\n\n", maxSteps);

}

}

int main() {

Member members[2] = {

{"shrushti", {12000, 9000, 15000, 8000, 10001, 11000, 7000, 6000, 12000, 9500,

13000, 8700, 10000, 10500, 9900, 12300, 5000, 8700, 10400, 9400,

8800, 9100, 12000, 10100, 8900, 11500, 12500, 13000, 7000, 9800}},

{"sindhu", {8000, 9500, 7000, 12000, 13000, 14000, 15000, 16000, 11000, 9000,

8700, 10500, 9800, 9600, 9400, 15000, 15500, 17000, 18000, 12000,

11000, 10000, 8900, 8700, 8500, 8400, 8300, 8200, 8100, 8000}}

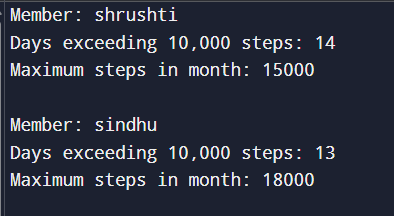
};

analyzeSteps(members, 2);

return 0;

}

OUTPUT-



QUESTIOM-02

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int songID;

struct Node\* next;

} Node;

Node\* createNode(int songID) {

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

newNode->songID = songID;

newNode->next = NULL;

return newNode;

}

Node\* reversePlaylistSegment(Node\* head, int m, int n) {

if (!head || m == n) return head;

Node dummy;

dummy.next = head;

Node\* prev = &dummy;

for (int i = 1; i < m; i++)

prev = prev->next;

Node\* start = prev->next;

Node\* then = start->next;

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

start->next = then->next;

then->next = prev->next;

prev->next = then;

then = start->next;

}

return dummy.next;

}

void printList(Node\* head) {

while (head) {

printf("%d", head->songID);

if (head->next) printf(" -> ");

head = head->next;

}

printf("\n");

}

int main() {

Node\* head = createNode(101);

head->next = createNode(102);

head->next->next = createNode(103);

head->next->next->next = createNode(104);

head->next->next->next->next = createNode(105);

head->next->next->next->next->next = createNode(106);

head->next->next->next->next->next->next = createNode(107);

printf("Original Playlist: ");

printList(head);

int m = 2, n = 5;

head = reversePlaylistSegment(head, m, n);

printf("Modified Playlist: ");

printList(head);

return 0;

}

OUTPUT-



QUESTION-03

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct Node {

char ch;

int freq;

struct Node \*left, \*right;

} Node;

typedef struct Queue {

Node\* arr[100];

int front, rear;

} Queue;

void enqueue(Queue\* q, Node\* node) {

q->arr[++q->rear] = node;

}

Node\* dequeue(Queue\* q) {

return q->arr[++q->front];

}

int isEmpty(Queue\* q) {

return q->front == q->rear;

}

Node\* createNode(char ch) {

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

node->ch = ch;

node->freq = 1;

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

return node;

}

void insert(Node\*\* root, char ch) {

if (\*root == NULL) {

\*root = createNode(ch);

return;

}

Queue q = {.front=-1, .rear=-1};

enqueue(&q, \*root);

while (!isEmpty(&q)) {

Node\* temp = dequeue(&q);

if (temp->ch == ch) {

temp->freq++;

return;

}

if (temp->left) enqueue(&q, temp->left);

if (temp->right) enqueue(&q, temp->right);

}

q.front = q.rear = -1;

enqueue(&q, \*root);

while (!isEmpty(&q)) {

Node\* temp = dequeue(&q);

if (!temp->left) {

temp->left = createNode(ch);

return;

} else enqueue(&q, temp->left);

if (!temp->right) {

temp->right = createNode(ch);

return;

} else enqueue(&q, temp->right);

}

}

void levelOrder(Node\* root) {

if (!root) return;

Queue q = {.front=-1, .rear=-1};

enqueue(&q, root);

while (!isEmpty(&q)) {

Node\* temp = dequeue(&q);

printf("(%c,%d) ", temp->ch, temp->freq);

if (temp->left) enqueue(&q, temp->left);

if (temp->right) enqueue(&q, temp->right);

}

}

int main() {

char str[] = "programming";

Node\* root = NULL;

for (int i = 0; i < strlen(str); i++)

insert(&root, str[i]);

printf("Level-order traversal with frequency:\n");

levelOrder(root);

return 0;

}

OUTPUT-

