ASSIGNMENT - 02

# Formative Assessment - LAB

## Section - 2

#include <stdio.h>  
  
#define MAX\_QUEUE 5  
  
int serviceLine[MAX\_QUEUE];  
int qStart = -1, qEnd = -1;  
  
int fullQueue() {  
 return (qStart == 0 && qEnd == MAX\_QUEUE - 1) || (qStart == qEnd + 1);  
}  
  
int emptyQueue() {  
 return qStart == -1;  
}  
  
void addCustomer(int token) {  
 if (fullQueue()) {  
 printf("Queue is full. Please wait.\n");  
 return;  
 }  
 if (qStart == -1) qStart = 0;  
 qEnd = (qEnd + 1) % MAX\_QUEUE;  
 serviceLine[qEnd] = token;  
 printf("Call added: %d\n", token);  
}  
  
void removeCustomer() {  
 if (emptyQueue()) {  
 printf("Queue is empty. No calls to remove.\n");  
 return;  
 }  
 printf("Removed customer: %d\n", serviceLine[qStart]);  
 if (qStart == qEnd) {  
 qStart = qEnd = -1;  
 } else {  
 qStart = (qStart + 1) % MAX\_QUEUE;  
 }  
}  
  
void showCustomers() {  
 if (emptyQueue()) {  
 printf("No customers in queue.\n");  
 return;  
 }  
 printf("Customers in queue: ");  
 int idx = qStart;  
 do {  
 printf("%d ", serviceLine[idx]);  
 idx = (idx + 1) % MAX\_QUEUE;  
 } while (idx != (qEnd + 1) % MAX\_QUEUE);  
 printf("\n");  
}  
  
int main() {  
 addCustomer(101);  
 addCustomer(102);  
 addCustomer(103);  
 addCustomer(104);  
 addCustomer(105);  
 showCustomers();  
  
 addCustomer(106);  
  
 removeCustomer();  
 removeCustomer();  
 showCustomers();  
  
 addCustomer(106);  
 addCustomer(107);  
 showCustomers();  
  
 return 0;  
}

## Section - 3

#include <stdio.h>  
  
#define STACK\_CAP 10  
  
int undoBuffer[STACK\_CAP];  
int redoBuffer[STACK\_CAP];  
int undoPointer = -1, redoPointer = -1;  
  
void pushAction(int action) {  
 if (undoPointer < STACK\_CAP - 1) {  
 undoBuffer[++undoPointer] = action;  
 redoPointer = -1;  
 printf("Performed operation: \"op%d\"\n", action);  
 }  
}  
  
void undoAction() {  
 if (undoPointer >= 0) {  
 int step = undoBuffer[undoPointer--];  
 redoBuffer[++redoPointer] = step;  
 if (undoPointer >= 0)  
 printf("Undone. Next Operation that can be undone is = \"op%d\"\n", undoBuffer[undoPointer]);  
 else  
 printf("Undone. No more operations to undo.\n");  
 } else {  
 printf("Nothing to undo.\n");  
 }  
}  
  
void redoAction() {  
 if (redoPointer >= 0) {  
 int step = redoBuffer[redoPointer--];  
 undoBuffer[++undoPointer] = step;  
 if (redoPointer >= 0)  
 printf("Redo completed. Next Operation that can be redone is = \"op%d\"\n", redoBuffer[redoPointer]);  
 else  
 printf("Redo completed. No more operations to redo.\n");  
 } else {  
 printf("Nothing to redo.\n");  
 }  
}  
  
int main() {  
 pushAction(1);  
 pushAction(2);  
 pushAction(3);  
 pushAction(4);  
  
 undoAction();  
 undoAction();  
  
 redoAction();  
  
 pushAction(5);  
  
 undoAction();  
 return 0;  
}

# Formative Assessment - Theory

## QUESTION - 01

#include <stdio.h>  
#include <string.h>  
  
#define MONTH\_DAYS 30  
  
typedef struct {  
 char memberName[50];  
 int stepsPerDay[MONTH\_DAYS];  
} StepTracker;  
  
void evaluateSteps(StepTracker records[], int total) {  
 for (int i = 0; i < total; i++) {  
 int activeDays = 0;  
 int bestDay = records[i].stepsPerDay[0];  
 for (int d = 0; d < MONTH\_DAYS; d++) {  
 if (records[i].stepsPerDay[d] > 10000)  
 activeDays++;  
 if (records[i].stepsPerDay[d] > bestDay)  
 bestDay = records[i].stepsPerDay[d];  
 }  
 printf("Member: %s\n", records[i].memberName);  
 printf("Days exceeding 10,000 steps: %d\n", activeDays);  
 printf("Maximum steps in month: %d\n\n", bestDay);  
 }  
}  
  
int main() {  
 StepTracker data[2] = {  
 {"ananya", {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}},  
 {"krishna", {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}}  
 };  
  
 evaluateSteps(data, 2);  
 return 0;  
}

## QUESTION - 02

#include <stdio.h>  
#include <stdlib.h>  
  
typedef struct Music {  
 int trackID;  
 struct Music\* next;  
} Music;  
  
Music\* createTrack(int id) {  
 Music\* node = (Music\*)malloc(sizeof(Music));  
 node->trackID = id;  
 node->next = NULL;  
 return node;  
}  
  
Music\* reverseSegment(Music\* head, int start, int end) {  
 if (!head || start == end) return head;  
  
 Music dummy;  
 dummy.next = head;  
 Music\* prev = &dummy;  
  
 for (int i = 1; i < start; i++) prev = prev->next;  
  
 Music\* curr = prev->next;  
 Music\* nextNode = curr->next;  
  
 for (int i = 0; i < end - start; i++) {  
 curr->next = nextNode->next;  
 nextNode->next = prev->next;  
 prev->next = nextNode;  
 nextNode = curr->next;  
 }  
  
 return dummy.next;  
}  
  
void printPlaylist(Music\* head) {  
 while (head) {  
 printf("%d", head->trackID);  
 if (head->next) printf(" -> ");  
 head = head->next;  
 }  
 printf("\n");  
}  
  
int main() {  
 Music\* head = createTrack(201);  
 head->next = createTrack(202);  
 head->next->next = createTrack(203);  
 head->next->next->next = createTrack(204);  
 head->next->next->next->next = createTrack(205);  
 head->next->next->next->next->next = createTrack(206);  
 head->next->next->next->next->next->next = createTrack(207);  
  
 printf("Original Playlist: ");  
 printPlaylist(head);  
  
 head = reverseSegment(head, 2, 5);  
  
 printf("Modified Playlist: ");  
 printPlaylist(head);  
  
 return 0;  
}

## QUESTION - 03

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
typedef struct CharNode {  
 char symbol;  
 int freq;  
 struct CharNode \*left, \*right;  
} CharNode;  
  
typedef struct NodeQueue {  
 CharNode\* array[100];  
 int front, rear;  
} NodeQueue;  
  
void enqueueNode(NodeQueue\* q, CharNode\* n) {  
 q->array[++q->rear] = n;  
}  
  
CharNode\* dequeueNode(NodeQueue\* q) {  
 return q->array[++q->front];  
}  
  
int isQueueEmpty(NodeQueue\* q) {  
 return q->front == q->rear;  
}  
  
CharNode\* makeCharNode(char c) {  
 CharNode\* n = (CharNode\*)malloc(sizeof(CharNode));  
 n->symbol = c;  
 n->freq = 1;  
 n->left = n->right = NULL;  
 return n;  
}  
  
void insertChar(CharNode\*\* root, char c) {  
 if (\*root == NULL) {  
 \*root = makeCharNode(c);  
 return;  
 }  
  
 NodeQueue q = {.front=-1, .rear=-1};  
 enqueueNode(&q, \*root);  
  
 while (!isQueueEmpty(&q)) {  
 CharNode\* tmp = dequeueNode(&q);  
 if (tmp->symbol == c) {  
 tmp->freq++;  
 return;  
 }  
 if (tmp->left) enqueueNode(&q, tmp->left);  
 if (tmp->right) enqueueNode(&q, tmp->right);  
 }  
  
 q.front = q.rear = -1;  
 enqueueNode(&q, \*root);  
  
 while (!isQueueEmpty(&q)) {  
 CharNode\* tmp = dequeueNode(&q);  
 if (!tmp->left) {  
 tmp->left = makeCharNode(c);  
 return;  
 } else enqueueNode(&q, tmp->left);  
  
 if (!tmp->right) {  
 tmp->right = makeCharNode(c);  
 return;  
 } else enqueueNode(&q, tmp->right);  
 }  
}  
  
void printLevel(CharNode\* root) {  
 if (!root) return;  
 NodeQueue q = {.front=-1, .rear=-1};  
 enqueueNode(&q, root);  
 while (!isQueueEmpty(&q)) {  
 CharNode\* temp = dequeueNode(&q);  
 printf("(%c,%d) ", temp->symbol, temp->freq);  
 if (temp->left) enqueueNode(&q, temp->left);  
 if (temp->right) enqueueNode(&q, temp->right);  
 }  
}  
  
int main() {  
 char word[] = "algorithm";  
 CharNode\* root = NULL;  
  
 for (int i = 0; i < strlen(word); i++)  
 insertChar(&root, word[i]);  
  
 printf("Level-order traversal with frequency:\n");  
 printLevel(root);  
  
 return 0;  
}