# Assignment - 2

**1. Write a C program to remove duplicate element from sorted Linked List.**

**Input:**

**2 -> 3 -> 3 -> 4**

**Output:**

**2 -> 3 -> 4**

**Soln:**

#include<stdio.h>

#include<stdlib.h>

struct Node {

int data;

struct Node\* next;

};

void removeDuplicates(struct Node\* head) {

struct Node\* current = head;

struct Node\* next\_next;

if (current == NULL)

return;

while (current->next != NULL) {

if (current->data == current->next->data) {

next\_next = current->next->next;

free(current->next);

current->next = next\_next;

} else {

current = current->next;

}

}

}

void push(struct Node\*\* head\_ref, int new\_data) {

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

new\_node->data = new\_data;

new\_node->next = (\*head\_ref);

(\*head\_ref) = new\_node;

}

void printList(struct Node\* node) {

while (node != NULL) {

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

node = node->next;

}

printf("NULL\n");

}

int main() {

struct Node\* head = NULL;

push(&head, 4);

push(&head, 3);

push(&head, 3);

push(&head, 2);

printf("Input: ");

printList(head);

removeDuplicates(head);

printf("Output: ");

printList(head);

return 0;

}

1. **Write a C program to rotate a doubly linked list by N nodes.**

**Input: (When N=2)**

**a b c d e**

**Output:**

**c d e a b**

**Input: (When N=4)**

**a b c d e f g h**

**Output:**

**e f g h a b c d**

**Soln:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

char data;

struct Node\* prev;

struct Node\* next;

};

struct Node\* createNode(char data) {

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

newNode->data = data;

newNode->prev = NULL;

newNode->next = NULL;

return newNode;

}

void insertEnd(struct Node\*\* head, char 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;

newNode->prev = temp;

}

void rotateByN(struct Node\*\* head, int N) {

if (N == 0)

return;

struct Node\* current = \*head;

int count = 1;

while (count < N && current != NULL) {

current = current->next;

count++;

}

if (current == NULL)

return;

struct Node\* NthNode = current;

while (current->next != NULL)

current = current->next;

current->next = \*head;

(\*head)->prev = current;

\*head = NthNode->next;

(\*head)->prev = NULL;

NthNode->next = NULL;

}

void printList(struct Node\* node) {

while (node != NULL) {

printf("%c ", node->data);

node = node->next;

}

printf("\n");

}

int main() {

struct Node\* head = NULL;

insertEnd(&head, 'a');

insertEnd(&head, 'b');

insertEnd(&head, 'c');

insertEnd(&head, 'd');

insertEnd(&head, 'e');

int N = 2;

printf("Input: ");

printList(head);

rotateByN(&head, N);

printf("Output: ");

printList(head);

return 0;

}

1. **Write a C program to sort the elements of a queue in ascending order.**

**Input**

**4 2 7 5 1**

**Output**

**1 2 4 5 7**

**Soln:**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 100

struct Queue {

int items[MAX\_SIZE];

int front;

int rear;

};

struct Queue\* createQueue() {

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

queue->front = -1;

queue->rear = -1;

return queue;

}

int isEmpty(struct Queue\* queue) {

if (queue->rear == -1)

return 1;

else

return 0;

}

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

if (queue->rear == MAX\_SIZE - 1)

printf("\nQueue is Full!!");

else {

if (queue->front == -1)

queue->front = 0;

queue->rear++;

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

}

}

int dequeue(struct Queue\* queue) {

int item;

if (isEmpty(queue)) {

printf("\nQueue is Empty!!");

item = -1;

} else {

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

queue->front++;

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

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

}

}

return item;

}

void sortQueue(struct Queue\* queue) {

int temp[MAX\_SIZE];

int i, j, size = 0;

while (!isEmpty(queue)) {

temp[size++] = dequeue(queue);

}

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

for (j = 0; j < size - i - 1; j++) {

if (temp[j] > temp[j + 1]) {

int t = temp[j];

temp[j] = temp[j + 1];

temp[j + 1] = t;

}

}

}

for (i = 0; i < size; i++) {

enqueue(queue, temp[i]);

}

}

void display(struct Queue\* queue) {

int i;

if (isEmpty(queue))

printf("\nQueue is Empty!!\n");

else {

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

for (i = queue->front; i < queue->rear + 1; i++) {

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

}

printf("\n");

}

}

int main() {

struct Queue\* queue = createQueue();

int n, i, data;

printf("Enter the number of elements in the queue: ");

scanf("%d", &n);

printf("Enter the elements of the queue:\n");

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

scanf("%d", &data);

enqueue(queue, data);

}

printf("\nOriginal queue:\n");

display(queue);

sortQueue(queue);

printf("\nQueue after sorting:\n");

display(queue);

return 0;

}

1. **List all queue function operations available for manipulation of data elements in c**

**Soln:**

Enqueue()

Dequeue()

isEmpty()

isFull()

Peek()

Size()

Front()

Rear()

Clear()

1. Reverse the given string using stack

Input: (string)

"LetsLearn"

Output: (string)

"nraeLsteL"

Soln:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_SIZE 100

struct Stack {

int top;

unsigned capacity;

char\* array;

};

struct Stack\* createStack(unsigned capacity) {

struct Stack\* stack = (struct Stack\*)malloc(sizeof(struct Stack));

stack->capacity = capacity;

stack->top = -1;

stack->array = (char\*)malloc(stack->capacity \* sizeof(char));

return stack;

}

int isFull(struct Stack\* stack) {

return stack->top == stack->capacity - 1;

}

int isEmpty(struct Stack\* stack) {

return stack->top == -1;

}

void push(struct Stack\* stack, char item) {

if (isFull(stack))

return;

stack->array[++stack->top] = item;

}

char pop(struct Stack\* stack) {

if (isEmpty(stack))

return '\0';

return stack->array[stack->top--];

}

void reverseString(char\* string) {

int length = strlen(string);

struct Stack\* stack = createStack(length);

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

push(stack, string[i]);

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

string[i] = pop(stack);

}

int main() {

char string[] = "LetsLearn";

printf("Original string: %s\n", string);

reverseString(string);

printf("Reversed string: %s\n", string);

return 0;

}

1. **Insert value in sorted way in a sorted doubly linked list. Given a sorted doubly linked list and a value to insert, write a function to insert the value in sorted way.**

**Soln:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* prev;

struct Node\* next;

};

struct Node\* newNode(int data) {

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

node->data = data;

node->prev = NULL;

node->next = NULL;

return node;

}

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

struct Node\* new\_node = newNode(data);

struct Node\* current;

if (\*head\_ref == NULL || (\*head\_ref)->data >= new\_node->data) {

new\_node->next = \*head\_ref;

if (\*head\_ref != NULL)

(\*head\_ref)->prev = new\_node;

\*head\_ref = new\_node;

} else {

current = \*head\_ref;

while (current->next != NULL && current->next->data < new\_node->data) {

current = current->next;

}

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

if (current->next != NULL)

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

current->next = new\_node;

new\_node->prev = current;

}

}

void printList(struct Node\* node) {

while (node != NULL) {

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

node = node->next;

}

printf("\n");

}

int main() {

struct Node\* head = NULL;

insertSorted(&head, 3);

insertSorted(&head, 2);

insertSorted(&head, 5);

insertSorted(&head, 8);

insertSorted(&head, 10);

insertSorted(&head, 12);

printf("Sorted Doubly Linked List: ");

printList(head);

return 0;

}

1. **Write a C program to insert/delete and count the number of elements in a queue.**

**Soln:**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 100

struct Queue {

int items[MAX\_SIZE];

int front;

int rear;

};

void initQueue(struct Queue\* q) {

q->front = -1;

q->rear = -1;

}

int isEmpty(struct Queue\* q) {

return (q->rear == -1);

}

int isFull(struct Queue\* q) {

return (q->rear == MAX\_SIZE - 1);

}

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

if (isFull(q)) {

printf("Queue Overflow!\n");

return;

}

if (isEmpty(q)) {

q->front = 0;

}

q->rear++;

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

}

int dequeue(struct Queue\* q) {

int item;

if (isEmpty(q)) {

printf("Queue Underflow!\n");

return -1;

}

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

q->front++;

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

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

}

return item;

}

int countElements(struct Queue\* q) {

if (isEmpty(q))

return 0;

else

return (q->rear - q->front + 1);

}

void printQueue(struct Queue\* q) {

int i;

if (isEmpty(q)) {

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

return;

}

printf("Queue elements are: ");

for (i = q->front; i <= q->rear; i++)

printf("%d ", q->items[i]);

printf("\n");

}

int main() {

struct Queue q;

initQueue(&q);

printf("Initialize a queue!\n");

printf("Check the queue is empty or not? %s\n", isEmpty(&q) ? "Yes" : "No");

printf("Number of elements in queue: %d\n", countElements(&q));

printf("Insert some elements into the queue:\n");

enqueue(&q, 1);

enqueue(&q, 2);

enqueue(&q, 3);

printQueue(&q);

printf("Number of elements in queue: %d\n", countElements(&q));

printf("Delete two elements from the said queue:\n");

dequeue(&q);

dequeue(&q);

printQueue(&q);

printf("Number of elements in queue: %d\n", countElements(&q));

printf("Insert another element into the queue:\n");

enqueue(&q, 4);

printQueue(&q);

printf("Number of elements in the queue: %d\n", countElements(&q));

return 0;

}

1. **Write a C program to Find whether an array is a subset of another array.**

Soln:

#include <stdio.h>

int isSubset(int arr1[], int m, int arr2[], int n) {

int i, j;

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

for (j = 0; j < m; j++) {

if (arr2[i] == arr1[j])

break;

}

if (j == m)

return 0;

}

return 1;

}

int main() {

int arr1[] = {11, 1, 13, 21, 3, 7};

int arr2[] = {11, 3, 7, 1};

int m = sizeof(arr1) / sizeof(arr1[0]);

int n = sizeof(arr2) / sizeof(arr2[0]);

if (isSubset(arr1, m, arr2, n))

printf("arr2[] is a subset of arr1[]\n");

else

printf("arr2[] is not a subset of arr1[]\n");

int arr3[] = {10, 5, 2, 23, 19};

int arr4[] = {19, 5, 3};

m = sizeof(arr3) / sizeof(arr3[0]);

n = sizeof(arr4) / sizeof(arr4[0]);

if (isSubset(arr3, m, arr4, n))

printf("arr4[] is a subset of arr3[]\n");

else

printf("arr4[] is not a subset of arr3[]\n");

return 0;

}