Week 2 Assignment

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

Input:

2 -> 3 -> 3 -> 4

Output:

2 -> 3 -> 4

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

void removeDuplicates(struct Node\* head) {

struct Node\* current = head;

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

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

struct Node\* temp = current->next;

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

free(temp);

} else {

current = current->next;

}

}

}

void printList(struct Node\* node) {

while (node != NULL) {

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

node = node->next;

}

printf("\n");

}

int main() {

struct Node\* head = NULL;

struct Node\* second = NULL;

struct Node\* third = NULL;

struct Node\* fourth = NULL;

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

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

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

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

head->data = 2;

head->next = second;

second->data = 3;

second->next = third;

third->data = 3;

third->next = fourth;

fourth->data = 4;

fourth->next = NULL;

printf("Original Linked List: ");

printList(head);

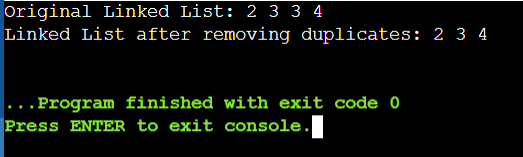
removeDuplicates(head);

printf("Linked List after removing duplicates: ");

printList(head);

return 0;

}



2. 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

#include <stdio.h>

#include <stdlib.h>

struct Node {

char data;

struct Node\* prev;

struct Node\* next;

};

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

if (\*head\_ref == NULL || N == 0)

return;

struct Node\* current = \*head\_ref;

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\_ref;

(\*head\_ref)->prev = current;

\*head\_ref = NthNode->next;

(\*head\_ref)->prev = NULL;

NthNode->next = NULL;

}

void insert(struct Node\*\* head\_ref, char new\_data) {

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

new\_node->data = new\_data;

new\_node->prev = NULL;

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

if ((\*head\_ref) != NULL)

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

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

}

void printList(struct Node\* node) {

while (node != NULL) {

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

node = node->next;

}

printf("\n");

}

int main() {

struct Node\* head = NULL;

insert(&head, 'a');

insert(&head, 'b');

insert(&head, 'c');

insert(&head, 'd');

insert(&head, 'e');

printf("Original Doubly Linked List: ");

printList(head);

int N = 2;

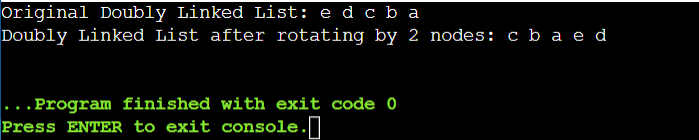
rotateByN(&head, N);

printf("Doubly Linked List after rotating by %d nodes: ", N);

printList(head);

return 0;

}



3. 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

#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("Queue 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 n = queue->rear - queue->front + 1;

int\* arr = (int\*)malloc(n \* sizeof(int));

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

arr[i] = dequeue(queue);

}

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

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

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

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

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

enqueue(queue, arr[i]);

}

}

void printQueue(struct Queue\* queue) {

int i = queue->front;

if (isEmpty(queue)) {

printf("Queue is empty");

} else {

while (i <= queue->rear) {

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

i++;

}

}

printf("\n");

}

int main() {

struct Queue\* queue = createQueue();

enqueue(queue, 4);

enqueue(queue, 2);

enqueue(queue, 7);

enqueue(queue, 5);

enqueue(queue, 1);

printf("Original Queue: ");

printQueue(queue);

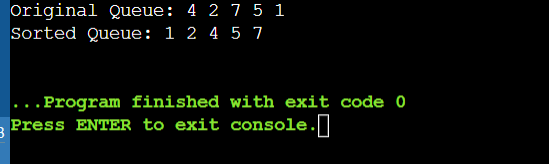
sortQueue(queue);

printf("Sorted Queue: ");

printQueue(queue);

return 0;

}



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

* Enqueue: Insert an element into the queue.
* Dequeue: Remove and return the front element from the queue.
* Front: Get the front element of the queue without removing it.
* IsEmpty: Check if the queue is empty.
* IsFull: Check if the queue is full.
* Size: Get the number of elements currently in the queue.
* Clear: Remove all elements from the queue.
* Peek: Similar to Front, get the front element of the queue without removing it. (Sometimes used interchangeably with Front)
* Display: Display all elements of the queue (used for debugging and visualization purposes).

5. Reverse the given string using stack

Input: (string)

"LetsLearn"

Output: (string)

"nraeLsteL"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_SIZE 100

struct Stack {

int top;

char items[MAX\_SIZE];

};

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

if (stack->top == MAX\_SIZE - 1) {

printf("Stack Overflow\n");

} else {

stack->top++;

stack->items[stack->top] = item;

}

}

char pop(struct Stack\* stack) {

if (stack->top == -1) {

printf("Stack Underflow\n");

return '\0';

} else {

char item = stack->items[stack->top];

stack->top--;

return item;

}

}

int isEmpty(struct Stack\* stack) {

return stack->top == -1;

}

void reverseString(char\* str) {

int length = strlen(str);

struct Stack stack;

stack.top = -1;

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

push(&stack, str[i]);

}

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

str[i] = pop(&stack);

}

}

int main() {

char str[] = "LetsLearn";

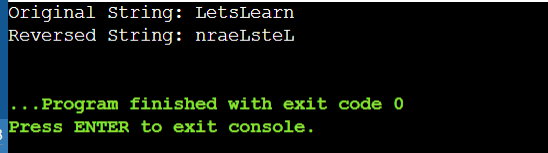
printf("Original String: %s\n", str);

reverseString(str);

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

return 0;

}



6. 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.

#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));

newNode->data = data;

newNode->prev = NULL;

newNode->next = NULL;

return newNode;

}

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

struct Node\* newNode = createNode(value);

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

newNode->next = \*head\_ref;

if (\*head\_ref != NULL)

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

\*head\_ref = newNode;

} else {

struct Node\* current = \*head\_ref;

while (current->next != NULL && current->next->data < value) {

current = current->next;

}

newNode->next = current->next;

if (current->next != NULL)

current->next->prev = newNode;

current->next = newNode;

newNode->prev = current;

}

}

void printList(struct Node\* head) {

struct Node\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

printf("\n");

}

int main() {

struct Node\* head = NULL;

insertSorted(&head, 3);

insertSorted(&head, 5);

insertSorted(&head, 8);

insertSorted(&head, 10);

insertSorted(&head, 12);

printf("Original Doubly Linked List: ");

printList(head);

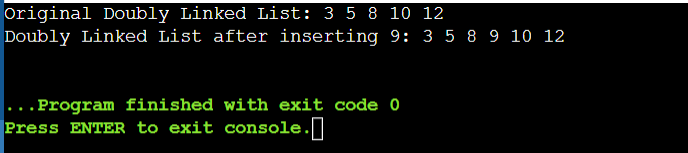
insertSorted(&head, 9);

printf("Doubly Linked List after inserting 9: ");

printList(head);

return 0;

}



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

Expected Output:

Initialize a queue!

Check the queue is empty or not? Yes

Number of elements in queue: 0

Insert some elements into the queue:

Queue elements are: 1 2 3

Number of elements in queue: 3

Delete two elements from the said queue:

Queue elements are: 3

Number of elements in queue: 1

Insert another element into the queue:

Queue elements are: 3 4

Number of elements in the queue: 2

#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) {

return queue->rear == -1;

}

int isFull(struct Queue\* queue) {

return queue->rear == MAX\_SIZE - 1;

}

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

if (isFull(queue)) {

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 (isEmpty(queue)) {

printf("Queue is empty\n");

return -1;

} else {

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

queue->front++;

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

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

}

return item;

}

}

int countElements(struct Queue\* queue) {

if (isEmpty(queue))

return 0;

else

return queue->rear - queue->front + 1;

}

void printQueue(struct Queue\* queue) {

if (isEmpty(queue)) {

printf("Queue is empty\n");

} else {

printf("Queue elements are: ");

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

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

}

printf("\n");

}

}

int main() {

struct Queue\* queue = createQueue();

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

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

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

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

enqueue(queue, 1);

enqueue(queue, 2);

enqueue(queue, 3);

printQueue(queue);

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

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

dequeue(queue);

dequeue(queue);

printQueue(queue);

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

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

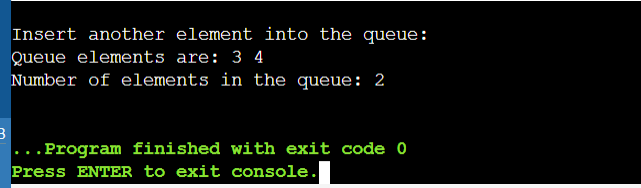
enqueue(queue, 4);

printQueue(queue);

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

return 0;

}



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

Input:

arr1[] = {11, 1, 13, 21, 3, 7}, arr2[] = {11, 3, 7, 1}

Output:

arr2[] is a subset of arr1[]

Input:

arr1[] = {10, 5, 2, 23, 19}, arr2[] = {19, 5, 3}

Output:

arr2[] is not a subset of arr1[]

#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");

return 0;

}

