Assignment - 2

Name: Vishnu Sankar J College: Mepco Schlenk Engineering College

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

#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\* head) {

struct Node\* current = head;

while (current != NULL) {

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

current = current->next;

}

printf("\n");

}

struct Node\* insert(struct Node\* head, int data) {

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

newNode->data = data;

newNode->next = NULL;

if (head == NULL) {

return newNode;

}

struct Node\* current = head;

while (current->next != NULL) {

current = current->next;

}

current->next = newNode;

return head;

}

int main() {

struct Node\* head = NULL;

head = insert(head, 2);

head = insert(head, 3);

head = insert(head, 3);

head = insert(head, 4);

printf("Linked List: ");

printList(head);

removeDuplicates(head);

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

printList(head);

while (head != NULL) {

struct Node\* temp = head;

head = head->next;

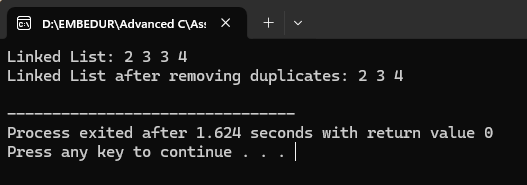
free(temp);

}

return 0;

}

Output:



---------------------------------------------------------------------------------------------------------------------------------------

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

Code:

#include <stdio.h>

#include <stdlib.h>

struct Node {

char data;

struct Node\* prev;

struct Node\* next;

};

struct Node\* rotateDoublyList(struct Node\* head, int N) {

if (head == NULL || N == 0) {

return head;

}

struct Node\* lastNode = head;

while (lastNode->next != NULL) {

lastNode = lastNode->next;

}

int i =0;

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

lastNode->next = head;

head->prev = lastNode;

head = head->next;

head->prev = NULL;

lastNode = lastNode->next;

lastNode->next = NULL;

}

return head;

}

void printList(struct Node\* head) {

while (head != NULL) {

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

head = head->next;

}

printf("\n");

}

struct Node\* insert(struct Node\* head, char data) {

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

newNode->data = data;

newNode->prev = NULL;

newNode->next = NULL;

if (head == NULL) {

return newNode;

}

struct Node\* current = head;

while (current->next != NULL) {

current = current->next;

}

current->next = newNode;

newNode->prev = current;

return head;

}

int main() {

struct Node\* head = NULL;

char data;

int N;

printf("Elements of the doubly linked list");

while (1) {

scanf(" %c", &data);

if (data == '0') {

break;

}

head = insert(head, data);

}

printf("Enter the number of nodes to rotate: ");

scanf("%d", &N);

printf("Doubly Linked List: ");

printList(head);

head = rotateDoublyList(head, N);

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

printList(head);

while (head != NULL) {

struct Node\* temp = head;

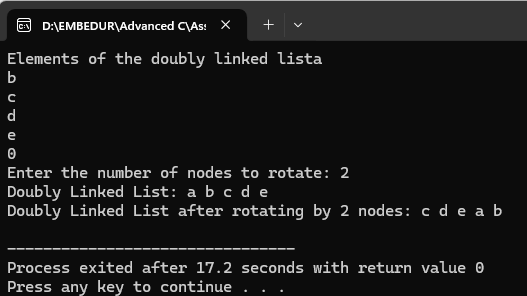
head = head->next;

free(temp);

}

return 0;

}



---------------------------------------------------------------------------------------------------------------------------------------

3)Write a C program to sort the elements of a queue in ascending order.

#include <stdio.h>

#define MAX\_SIZE 100

int queue[MAX\_SIZE];

int front = -1, back = -1;

void enqueue(int item) {

if (back == MAX\_SIZE - 1) {

printf("Error: Queue is full\n");

return;

}

if (front == -1) {

front = 0;

}

back++;

queue[back] = item;

}

int dequeue() {

if (front == -1 || front > back) {

printf("Error: Queue is empty\n");

return -1;

}

int item = queue[front];

front++;

return item;

}

void display() {

if (front == -1) {

printf("Error: Queue is empty\n");

return;

}

int i;

for (i = front; i <= back; i++) {

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

}

printf("\n");

}

void sort\_queue\_asc() {

int i, j, temp;

int n = back - front + 1;

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

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

if (queue[i] > queue[j]) {

temp = queue[i];

queue[i] = queue[j];

queue[j] = temp;

}

}

}

}

int main() {

enqueue(4);

enqueue(2);

enqueue(7);

enqueue(5);

enqueue(1);

printf("\nElements of the queue:\n");

display();

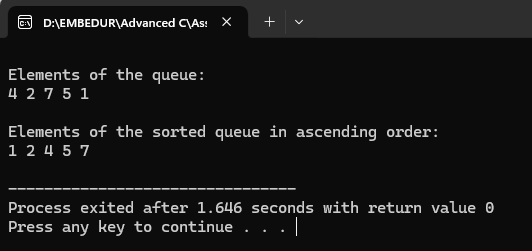
sort\_queue\_asc();

printf("\nElements of the sorted queue in ascending order:\n");

display();

return 0;

}



---------------------------------------------------------------------------------------------------------------------------------------

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

#include <stdio.h>

#define SIZE\_OF\_QUEUE 7

void enQueue(int);

void deQueue();

void display();

int array\_of\_Queue[SIZE\_OF\_QUEUE], front\_index = -1, rear\_index = -1;

int main() {

int data;

char ch;

do

{

int choice;

scanf("%d",&choice);

switch (choice)

{

case 1 :

printf("\nEnter the value to be inserted\n");

scanf("%d",&data);

enQueue(data);

break;

case 2 :

printf("\nContents of the Queue are::\n");

display();

break;

case 3 :

printf("\nDequeue Done.\n");

deQueue();

break;

default :

printf("Wrong Entry\n");

break;

}

printf("\nDo you want to continue (Type y or n)\n");

scanf(" %c",&ch);

} while (ch == 'Y'|| ch == 'y');

return 0;

}

void enQueue(int value) {

if (rear\_index == SIZE\_OF\_QUEUE - 1)

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

else {

if (front\_index == -1)

front\_index = 0;

rear\_index++;

array\_of\_Queue[rear\_index] = value;

printf("\nInserted -> %d", value);

}

}

void deQueue() {

if (front\_index == -1)

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

else {

printf("\nDeleted : %d", array\_of\_Queue[front\_index]);

front\_index++;

if (front\_index > rear\_index)

front\_index = rear\_index = -1;

}

}

void display() {

if (rear\_index == -1)

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

else {

int i;

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

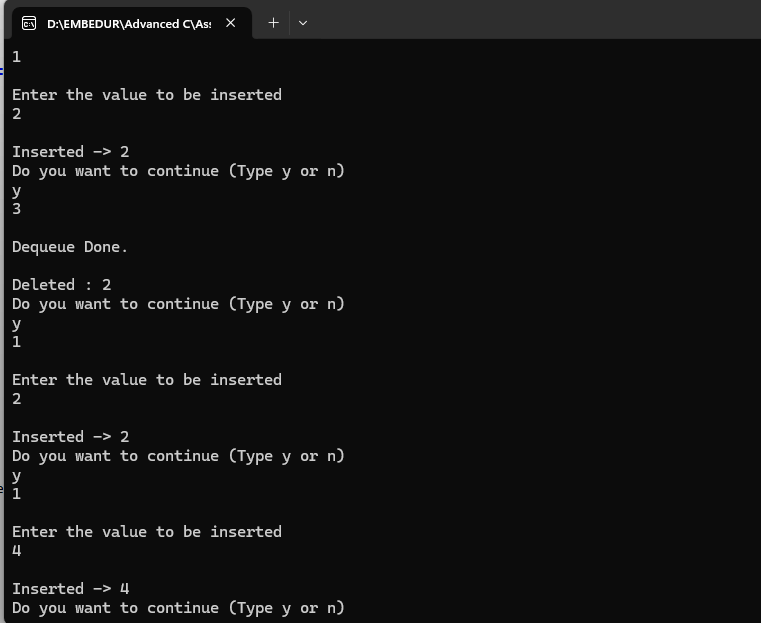
for (i = front\_index; i <= rear\_index; i++)

printf("%d ", array\_of\_Queue[i]);

}

printf("\n");

}



---------------------------------------------------------------------------------------------------------------------------------------

5)Reverse the given string using stack

Input: (string)

"LetsLearn"

Output: (string)

"nraeLsteL"

Code:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_SIZE 100

struct Stack {

char data[MAX\_SIZE];

int top;

};

void initializeStack(struct Stack\* stack) {

stack->top = -1;

}

int isEmpty(struct Stack\* stack) {

return stack->top == -1;

}

int isFull(struct Stack\* stack) {

return stack->top == MAX\_SIZE - 1;

}

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

if (isFull(stack)) {

printf("Stack is full. Cannot push.\n");

exit(EXIT\_FAILURE);

}

stack->data[++(stack->top)] = character;

}

char pop(struct Stack\* stack) {

if (isEmpty(stack)) {

printf("Stack is empty. Cannot pop.\n");

exit(EXIT\_FAILURE);

}

return stack->data[(stack->top)--];

}

void reverseString(char\* input) {

struct Stack charStack;

initializeStack(&charStack);

int i;

for (i = 0; i < strlen(input); ++i) {

push(&charStack, input[i]);

}

for (i = 0; i < strlen(input); ++i) {

input[i] = pop(&charStack);

}

}

int main() {

char input[] = "LetsLearn";

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

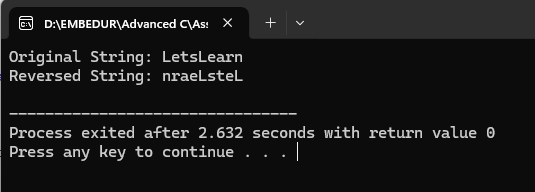
reverseString(input);

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

return 0;

}

Output:



---------------------------------------------------------------------------------------------------------------------------------------

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;

}

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

struct Node\* newNode = createNode(value);

if (head == NULL || value <= head->data) {

newNode->next = head;

if (head != NULL) {

head->prev = newNode;

}

return newNode;

}

struct Node\* current = head;

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

current = current->next;

}

newNode->next = current->next;

newNode->prev = current;

if (current->next != NULL) {

current->next->prev = newNode;

}

current->next = newNode;

return head;

}

void printList(struct Node\* head) {

while (head != NULL) {

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

head = head->next;

}

printf("\n");

}

void freeList(struct Node\* head) {

struct Node\* temp;

while (head != NULL) {

temp = head;

head = head->next;

free(temp);

}

}

int main() {

struct Node\* head = NULL;

head = insertSorted(head, 10);

head = insertSorted(head, 20);

head = insertSorted(head, 30);

head = insertSorted(head, 25);

head = insertSorted(head, 15);

printf("Sorted Doubly Linked List: ");

printList(head);

int newValue = 22;

printf("\nInserting %d in the middle.\n", newValue);

head = insertSorted(head, newValue);

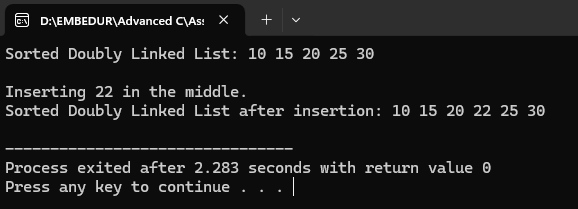
printf("Sorted Doubly Linked List after insertion: ");

printList(head);

freeList(head);

return 0;

}



---------------------------------------------------------------------------------------------------------------------------------------

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

Code:

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 5

struct Queue {

int front, rear;

int elements[MAX\_SIZE];

};

void initializeQueue(struct Queue \*queue) {

queue->front = -1;

queue->rear = -1;

}

int isEmpty(struct Queue \*queue) {

return (queue->front == -1 && queue->rear == -1);

}

int isFull(struct Queue \*queue) {

return (queue->rear + 1) % MAX\_SIZE == queue->front;

}

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

if (isFull(queue)) {

printf("Queue is full. Cannot insert %d.\n", value);

return;

}

if (isEmpty(queue)) {

queue->front = 0;

queue->rear = 0;

} else {

queue->rear = (queue->rear + 1) % MAX\_SIZE;

}

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

}

void dequeue(struct Queue \*queue) {

if (isEmpty(queue)) {

printf("Queue is empty. Cannot dequeue.\n");

return;

}

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

queue->front = -1;

queue->rear = -1;

} else {

queue->front = (queue->front + 1) % MAX\_SIZE;

}

}

int countElements(struct Queue \*queue) {

if (isEmpty(queue)) {

return 0;

} else {

return (MAX\_SIZE + queue->rear - queue->front + 1) % MAX\_SIZE;

}

}

void displayQueue(struct Queue \*queue) {

if (isEmpty(queue)) {

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

return;

}

int i = queue->front;

while (i != queue->rear) {

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

i = (i + 1) % MAX\_SIZE;

}

printf("%d\n", queue->elements[i]);

}

int main() {

struct Queue queue;

initializeQueue(&queue);

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

enqueue(&queue, 1);

enqueue(&queue, 2);

enqueue(&queue, 3);

printf("Queue elements are: ");

displayQueue(&queue);

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

dequeue(&queue);

printf("Queue elements are: ");

displayQueue(&queue);

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

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

enqueue(&queue, 4);

printf("Queue elements are: ");

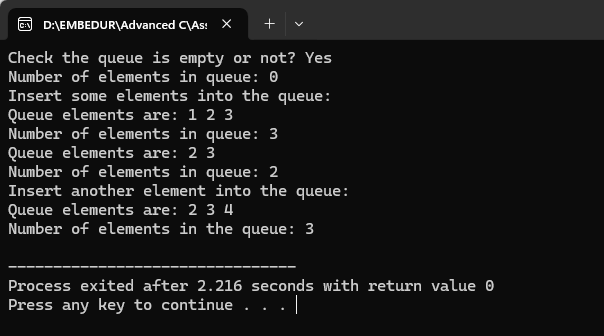
displayQueue(&queue);

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

return 0;

}

Output:



---------------------------------------------------------------------------------------------------------------------------------------

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

#include <stdio.h>

int isSubset(int array1[], int size1, int array2[], int size2) {

int i, j;

for (i = 0; i < size1; i++) {

int found = 0;

for (j = 0; j < size2; j++) {

if (array1[i] == array2[j]) {

found = 1;

break;

}

}

if (!found) {

return 0; }

}

return 1;

}

int main() {

int size1, size2;

printf("Enter the size of array1: ");

scanf("%d", &size1);

printf("Enter the size of array2: ");

scanf("%d", &size2);

int array1[size1], array2[size2];

printf("Enter elements of array1:\n");

for (int i = 0; i < size1; i++) {

printf("Element %d: ", i + 1);

scanf("%d", &array1[i]);

}

printf("Enter elements of array2:\n");

for (int i = 0; i < size2; i++) {

printf("Element %d: ", i + 1);

scanf("%d", &array2[i]);

}

if (isSubset(array1, size1, array2, size2)) {

printf("Array1 is a subset of Array2.\n");

} else {

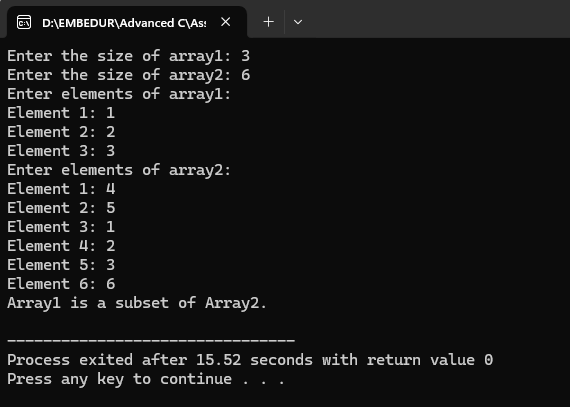
printf("Array1 is not a subset of Array2.\n");

}

return 0;

}

Output:



---------------------------------------------------------------------------------------------------------------------------------------