Week 7

Reverse, Sort and Concatenate 2 lists:

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* createNode(int data) {

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

if (newNode == NULL) {

printf("Memory allocation failed.\n");

exit(1);

}

newNode->data = data;

newNode->next = NULL;

return newNode;

}

void append(struct Node\*\* head, int 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;

}

void display(struct Node\* head) {

if (head == NULL) {

printf("Linked list is empty.\n");

return;

}

struct Node\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

}

void reverse(struct Node\*\* head) {

struct Node\* prev = NULL;

struct Node\* current = \*head;

struct Node\* next = NULL;

while (current != NULL) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

\*head = prev;

}

void bubbleSort(struct Node\* head) {

if (head == NULL) return;

struct Node\* current;

struct Node\* lastSorted = NULL;

int swapped;

do {

swapped = 0;

current = head;

while (current->next != lastSorted) {

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

int temp = current->data;

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

current->next->data = temp;

swapped = 1;

}

current = current->next;

}

lastSorted = current;

} while (swapped);

}

void concatenate(struct Node\*\* head1, struct Node\*\* head2) {

if (\*head1 == NULL) {

\*head1 = \*head2;

return;

}

struct Node\* temp = \*head1;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = \*head2;

}

void freeList(struct Node\* head) {

struct Node\* temp;

while (head != NULL) {

temp = head;

head = head->next;

free(temp);

}

}

int main() {

struct Node\* list1 = NULL;

struct Node\* list2 = NULL;

int ch, choice = 1;

printf("Ruqaiyya Mahreen 1BM23EE044");

while (1) {

printf("\nEnter 1. Create 2. Append 3. Reverse 4. Sort 5. Concatenate 6. Display 7. Exit: ");

scanf("%d", &ch);

switch (ch) {

case 1:

printf("Enter data for list1: ");

int data1;

scanf("%d", &data1);

list1 = createNode(data1);

break;

case 2:

printf("Enter data to append to list1: ");

int data2;

scanf("%d", &data2);

append(&list1, data2);

break;

case 3:

reverse(&list1);

printf("List1 after reversing:\n");

display(list1);

break;

case 4:

bubbleSort(list1);

printf("List1 after sorting:\n");

display(list1);

break;

case 5:

printf("Creating second list (list2):\n");

list2 = NULL;

do {

printf("Enter data to append to list2: ");

int data3;

scanf("%d", &data3);

append(&list2, data3);

printf("Do you want to add another element to list2? (0 or 1): ");

scanf("%d", &choice);

} while (choice == 1);

concatenate(&list1, &list2);

printf("List1 after concatenation with list2:\n");

display(list1);

break;

case 6:

printf("Displaying list1:\n");

display(list1);

break;

case 7:

freeList(list1);

freeList(list2);

printf("Exiting...\n");

exit(0);

default:

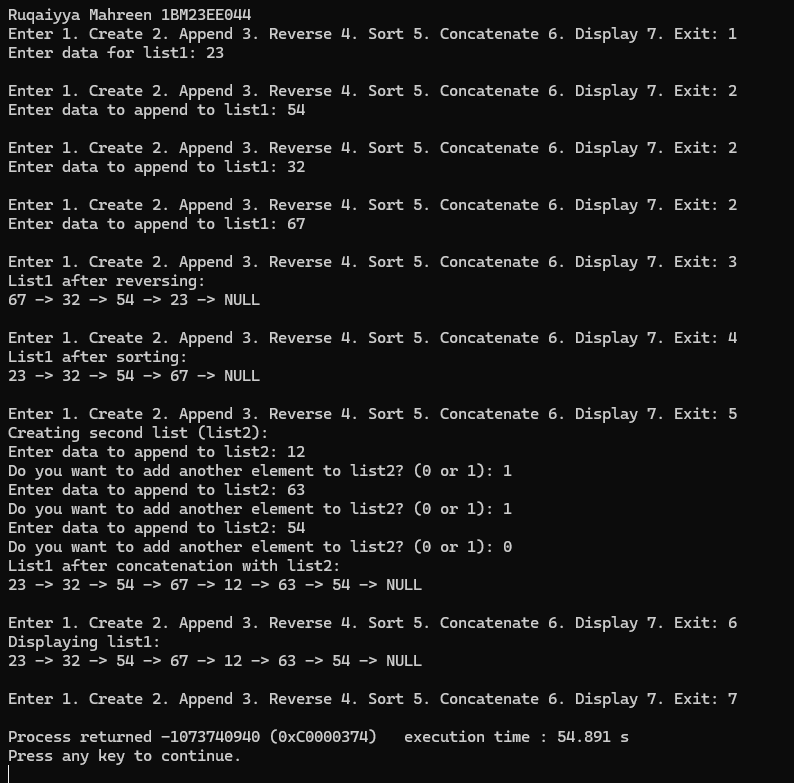
printf("Wrong choice!\n");

}

}

return 0;

}



Stack and Queue implementation using Singly Linked List:

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* createNode(int data) {

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

newNode->data = data;

newNode->next = NULL;

return newNode;

}

void push(struct Node\*\* top, int data) {

struct Node\* newNode = createNode(data);

newNode->next = \*top;

\*top = newNode;

}

int pop(struct Node\*\* top) {

if (\*top == NULL) {

printf("Stack underflow.\n");

return -1;

}

struct Node\* temp = \*top;

int poppedData = temp->data;

\*top = (\*top)->next;

free(temp);

return poppedData;

}

void displayStack(struct Node\* top) {

if (top == NULL) {

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

return;

}

printf("Stack: ");

struct Node\* temp = top;

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

}

void enqueue(struct Node\*\* front, struct Node\*\* rear, int data) {

struct Node\* newNode = createNode(data);

if (\*rear == NULL) {

\*front = \*rear = newNode;

return;

}

(\*rear)->next = newNode;

\*rear = newNode;

}

int dequeue(struct Node\*\* front, struct Node\*\* rear) {

if (\*front == NULL) {

printf("Queue underflow.\n");

return -1;

}

struct Node\* temp = \*front;

int dequeuedData = temp->data;

\*front = (\*front)->next;

if (\*front == NULL) {

\*rear = NULL;

}

free(temp);

return dequeuedData;

}

void displayQueue(struct Node\* front) {

if (front == NULL) {

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

return;

}

printf("Queue: ");

struct Node\* temp = front;

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

}

int main() {

struct Node\* stack = NULL;

struct Node\* queueFront = NULL;

struct Node\* queueRear = NULL;

int ch, data;

printf("Ruqaiyya Mahreen 1BM23EE044\n");

printf("Enter 1 for stack and 2 for queue: ");

scanf("%d", &ch);

switch (ch) {

case 1:

while (1) {

printf("\nStack Operations:\n");

printf("1. Push\n");

printf("2. Pop\n");

printf("3. Display\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &data);

switch (data) {

case 1:

printf("Enter data to push: ");

scanf("%d", &data);

push(&stack, data);

break;

case 2:

data = pop(&stack);

if (data != -1) {

printf("Popped value: %d\n", data);

}

break;

case 3:

displayStack(stack);

break;

case 4:

printf("Exiting stack operations.\n");

return 0;

default:

printf("Invalid choice!\n");

break;

}

}

break;

case 2:

while (1) {

printf("\nQueue Operations:\n");

printf("1. Enqueue\n");

printf("2. Dequeue\n");

printf("3. Display\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &data);

switch (data) {

case 1:

printf("Enter data to enqueue: ");

scanf("%d", &data);

enqueue(&queueFront, &queueRear, data);

break;

case 2:

data = dequeue(&queueFront, &queueRear);

if (data != -1) {

printf("Dequeued value: %d\n", data);

}

break;

case 3:

displayQueue(queueFront);

break;

case 4:

printf("Exiting queue operations.\n");

return 0;

default:

printf("Invalid choice!\n");

break;

}

}

break;

default:

printf("Wrong choice!\n");

break;

}

return 0;

}

