1. a) WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists.

Program:

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

{

struct Node\* newNode = createNode(data); if (\*head == NULL)

{

\*head = newNode;

} else

{

struct Node\* temp = \*head; while (temp->next != NULL)

{

temp = temp->next;

}

temp->next = newNode;

}

}

void printList(struct Node\* head)

{

struct Node\* temp = head; while (temp != NULL)

{

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

}

printf("NULL\n");

}

void sortList(struct Node\* head) {

if (head == NULL) return;

struct Node \*i, \*j; int temp;

for (i = head; i != NULL; i = i->next) {

for (j = i->next; j != NULL; j = j->next) { if (i->data > j->data) {

temp = i->data;

i->data = j->data;

j->data = temp;

}

}

}

}

void reverseList(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 concatenateLists(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;

}

int main() {

struct Node\* list1 = NULL; struct Node\* list2 = NULL;

int choice, data;

while (1) {

printf("\n1. Insert into List 1\n"); printf("2. Insert into List 2\n"); printf("3. Sort List 1\n"); printf("4. Reverse List 1\n");

printf("5. Concatenate List 1 and List 2\n"); printf("6. Print List 1\n");

printf("7. Print List 2\n"); printf("8. Exit\n"); printf("Enter your choice: "); scanf("%d", &choice);

switch (choice) { case 1:

printf("Enter data to insert into List 1: "); scanf("%d", &data);

insert(&list1, data); break;

case 2:

printf("Enter data to insert into List 2: "); scanf("%d", &data);

insert(&list2, data); break;

case 3:

sortList(list1); printf("List 1 sorted.\n"); break;

case 4:

reverseList(&list1); printf("List 1 reversed.\n"); break;

case 5:

concatenateLists(&list1, list2); printf("List 2 concatenated to List 1.\n"); break;

case 6:

printf("List 1: "); printList(list1); break;

case 7:

printf("List 2: "); printList(list2); break;

case 8:

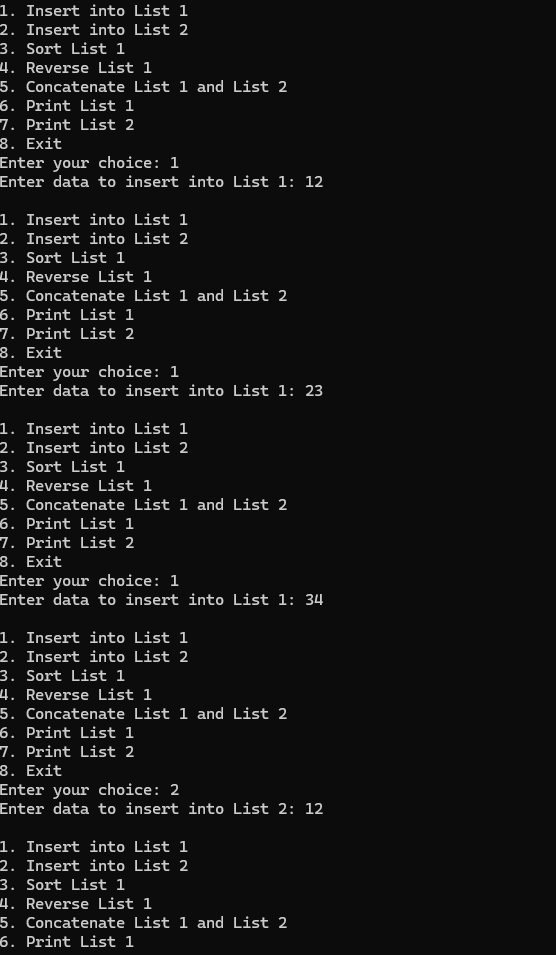
exit(0); default:

printf("Invalid choice! Please try again.\n");

}

}

return 0;

}

b) WAP to Implement Single Link List to simulate Stack & Queue Operations.

Program:

#include <stdio.h> #include <stdlib.h>

struct node { int value;

struct node \*next;

};

typedef struct node \*NODE;

NODE get\_node() {

NODE ptr = (NODE)malloc(sizeof(struct node)); if (ptr == NULL) {

printf("Memory not allocated\n");

}

return ptr;

}

NODE delete\_first(NODE first){ NODE temp=first;

if (first == NULL) { printf("Empty\n"); return NULL;

}

first=first->next; free(temp); return first;

}

NODE insert\_beginning(NODE first, int item) { NODE new\_node = get\_node();

new\_node->value = item; new\_node->next = first; return new\_node;

}

NODE insert\_end(NODE first, int item) { NODE new\_node = get\_node(); new\_node->value = item;

new\_node->next = NULL; if (first == NULL) {

return new\_node;

}

NODE temp = first;

while (temp->next != NULL) { temp = temp->next;

}

temp->next = new\_node; return first;

}

void display(NODE first) { NODE temp = first;

if (first == NULL) { printf("Empty\n"); return;

}

while (temp != NULL) { printf("%d ", temp->value); temp = temp->next;

}

printf("\n");

}

int main() {

int item, choice, deleted\_item; NODE first = NULL;

printf("Choose:\n"); printf("1. Stack\n"); printf("2. Queue\n"); printf("Enter choice (1/2): "); scanf("%d", &choice);

if (choice == 1) { while (1) {

printf("\nStack Operations:\n"); printf("1. Push\n");

printf("2. Pop\n"); printf("3. Display stack\n");

printf("4. Exit\n"); printf("Enter choice: "); scanf("%d", &choice);

switch (choice) { case 1:

printf("Enter item to push: "); scanf("%d", &item);

first = insert\_beginning(first, item); break;

case 2:

if (first != NULL) { deleted\_item = first->value; first = delete\_first(first);

printf("Deleted item from stack: %d\n", deleted\_item);

} else {

printf("Stack is empty\n");

}

break; case 3:

printf("Stack: "); display(first); break;

case 4:

exit(0); default:

printf("Invalid choice.\n");

}

}

}

else if (choice == 2) { while (1) {

printf("\nQueue Operations:\n"); printf("1. Insert\n");

printf("2. Delete\n"); printf("3. Display queue\n"); printf("4. Exit\n"); printf("Enter choice: "); scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter item to insert: "); scanf("%d", &item);

first = insert\_end(first, item); break;

case 2:

if (first != NULL) { deleted\_item = first->value; first = delete\_first(first);

printf("Deleted item from queue: %d\n", deleted\_item);

} else {

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

}

break; case 3:

printf("Queue: "); display(first); break;

case 4:

exit(0); default:

printf("Invalid choice.\n");

}

}

}

else {

printf("Invalid operation.\n");

}

return 0;

}

