**Single Linked Lists: Sort, Reverse, Concatenate, Stack & Queue implementation**

Program:

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

struct node {

int data;

struct node \*next;

};

struct node \*head = NULL;

struct node\* concatenate(struct node \*t1, struct node \*t2);

struct node\* reverse(struct node \*s);

void sort(struct node \*s);

void pop();

void push();

void enqueue();

void dequeue();

struct node\* createnode();

void display(struct node \*head);

struct node\* concatenate(struct node \*t1, struct node \*t2) {

if (t1 == NULL) return t2;

if (t2 == NULL) return t1;

struct node \*ptr = t1;

while (ptr->next != NULL) {

ptr = ptr->next;

}

ptr->next = t2;

return t1;

}

struct node\* reverse(struct node \*s) {

struct node \*prev = NULL, \*current = s, \*next = NULL;

while (current != NULL) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

return prev;

}

void sort(struct node \*s) {

if (s == NULL) {

printf("List is empty, nothing to sort.\n");

return;

}

struct node \*i, \*j;

int temp;

for (i = s; 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 push() {

struct node \*ptr = createnode();

ptr->next = head;

head = ptr;

}

void pop() {

if (head == NULL) {

printf("Stack is empty\n");

return;

}

struct node \*ptr = head;

head = head->next;

printf("Element %d removed\n", ptr->data);

free(ptr);

}

void enqueue() {

struct node \*ptr = createnode();

if (head == NULL) {

head = ptr;

} else {

struct node \*temp = head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = ptr;

}

}

void dequeue() {

if (head == NULL) {

printf("Queue is empty\n");

return;

}

struct node \*ptr = head;

head = head->next;

printf("Deleted element: %d\n", ptr->data);

free(ptr);

}

struct node\* createnode() {

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

if (newnode == NULL) {

printf("Memory allocation failed\n");

exit(1);

}

printf("Enter value to be inserted:\n");

scanf("%d", &newnode->data);

newnode->next = NULL;

return newnode;

}

void display(struct node \*head) {

if (head == NULL) {

printf("List is empty\n");

return;

}

struct node \*ptr = head;

while (ptr != NULL) {

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

ptr = ptr->next;

}

printf("NULL\n");

}

int main() {

int choice;

struct node \*ptr1 = NULL, \*ptr2 = NULL;

while (1) {

printf("\nEnter your choice:\n");

printf("1. Sort\n2. Concatenate\n3. Reverse\n4. Push into stack\n5. Pop out of stack\n6. Enqueue\n7. Dequeue\n8. Display\n9. Exit\n");

scanf("%d", &choice);

switch (choice) {

case 1:

if (head == NULL) {

printf("Create a list first.\n");

head = createnode();

}

sort(head);

printf("Sorted list: ");

display(head);

break;

case 2:

printf("Create first list:\n");

ptr1 = createnode();

printf("Create second list:\n");

ptr2 = createnode();

head = concatenate(ptr1, ptr2);

printf("Concatenated list: ");

display(head);

break;

case 3:

head = reverse(head);

printf("Reversed list: ");

display(head);

break;

case 4:

push();

break;

case 5:

pop();

break;

case 6:

enqueue();

break;

case 7:

dequeue();

break;

case 8:

display(head);

break;

case 9:

exit(0);

default:

printf("Invalid choice\n");

}

}

return 0;

}

Output:























