NAME: TASMIYA FATHIMA

USN: 1BM19CS172

**LAB 7: SINGLY LINKED LIST OPERATIONS: SORTING; REVERSING; CONCATENATION; STACK AND QUEUE IMPLEMENTATION**

QUESTION:

WAP Implement Single Link List with following operations a) Sort the linked list. b) Reverse the linked list. c) Concatenation of two linked lists d) implement Stack & Queues using Linked Representation

CODE:

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

typedef struct node{

int info;

struct node \*link;

}NODE;

NODE\* create(NODE \*start)

{

NODE \*temp1,\*temp2;

int n,i;

printf("Enter number of nodes:\n");

scanf("%d",&n);

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

{

temp1=(NODE\*)malloc(sizeof(NODE));

printf("Enter value for node %d:\n",i+1);

scanf("%d",&temp1->info);

temp1->link=NULL;

if(start==NULL)

start=temp1;

else

{

temp2=start;

while(temp2->link!=NULL)

temp2=temp2->link;

temp2->link=temp1;

}

}

return start;

}

NODE\* sort(NODE \*s)

{

NODE \*t1,\*t2;

int temp;

for(t1=s;t1->link!=NULL;t1=t1->link)

{

for(t2=t1->link;t2!=NULL;t2=t2->link)

{

if(t1->info>t2->info)

{

temp=t1->info;

t1->info=t2->info;

t2->info=temp;

}

}

}

printf("List has been sorted\n");

return s;

}

NODE\* rev(NODE \*start)

{

NODE \*t1,\*t2,\*s;

for(t1=start;t1->link!=NULL;t1=t1->link);

s=t1;

while(t1!=start)

{

for(t2=start;t2->link!=t1;t2=t2->link);

t1->link=t2;

t1=t2;

}

t1->link=NULL;

printf("List has been reversed\n");

return s;

}

NODE\* concat(NODE \*s1,NODE \*s2)

{

NODE \*t;

t=s1;

while(t->link!=NULL)

t=t->link;

t->link=s2;

printf("Two lists have been concatenated\n");

return s1;

}

void disp(NODE \*start)

{

NODE \*t;

printf("Elements of the list:\n");

for(t=start;t!=NULL;t=t->link)

{

if(t->link!=NULL)

printf("%d ",t->info);

else

printf("%d",t->info);

}

printf("\n");

}

void push();

void pop();

void display\_S();

struct node \*head;

void push ()

{

int info;

NODE \*ptr = (NODE\*)malloc(sizeof(NODE));

if(ptr == NULL)

{

printf("Stack Empty!\n");

}

else

{

printf("Enter the value:\n ");

scanf("%d",&info);

if(head==NULL)

{

ptr->info = info;

ptr -> link = NULL;

head=ptr;

}

else

{

ptr->info = info;

ptr->link = head;

head=ptr;

}

printf("Item is pushed\n");

}

}

void pop()

{

int item;

NODE \*ptr;

if (head == NULL)

{

printf("Stack Underflow\n");

}

else

{

item = head->info;

ptr = head;

head = head->link;

free(ptr);

printf("Item popped\n ");

}

}

void display\_S()

{

int i;

NODE \*ptr;

ptr=head;

if(ptr == NULL)

{

printf("Stack is empty\n ");

}

else

{

printf("The Stack elements are: \n");

while(ptr!=NULL)

{

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

ptr = ptr->link;

}

}

}

void enQueue();

void deQueue();

void display\_Q();

struct node \*front;

struct node \*rear;

void enQueue()

{

NODE \*ptr;

int item;

ptr = (NODE \*) malloc (sizeof(NODE));

if(ptr == NULL)

{

printf("Queue Overflow\n");

return;

}

else

{

printf("Enter value\n");

scanf("%d",&item);

ptr -> info = item;

if(front == NULL)

{

front = ptr;

rear = ptr;

front -> link = NULL;

rear -> link = NULL;

}

else

{

rear -> link = ptr;

rear = ptr;

rear->link = NULL;

}

}

}

void deQueue()

{

NODE \*ptr;

if(front == NULL)

{

printf("Queue Underflow\n");

return;

}

else

{

ptr = front;

front = front -> link;

free(ptr);

printf("Item deleted\n ");

}

}

void display\_Q()

{

NODE \*ptr;

ptr = front;

if(front == NULL)

{

printf("Queue is Empty\n");

}

else

{ printf("The Queue elements are:\n");

while(ptr != NULL)

{

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

ptr = ptr -> link;

}

}

}

void main()

{

NODE \*s1,\*s2;

int ch;

s1=NULL;

s2=NULL;

while(1)

{

printf("\n\*\*\*MENU\*\*\*\n");

printf("\n1.Sort\n2.Reverse\n3.Concatenate\n4.Display\n5.Stack (Push)\n6.Stack (Pop)\n7.Stack (Display)\n8.Queue (Enqueue)\n9.Queue (Dequeue)\n10.Queue (Display)\n11.Exit\n");

printf("Enter choice:\n");

scanf("%d",&ch);

switch(ch)

{

case 1: s1=create(s1);

s1=sort(s1);

break;

case 2: if(s1==NULL)

{

s1=create(s1);

}

s1=rev(s1);

break;

case 3: if(s1==NULL)

{

s1=create(s1);

}

s2=create(s2);

s1=concat(s1,s2);

break;

case 4: disp(s1);

break;

case 5: push();

break;

case 6: pop();

break;

case 7: display\_S();

break;

case 8: enQueue();

break;

case 9: deQueue();

break;

case 10: display\_Q();

break;

case 11: exit(0);

break;

default: printf("Wrong choice!");

break;

}

}

}

OUTPUT:



































