Lab-7,8

```
/*
WAP Implement Single Link List with following operations
a) Sort the linked list.
b) Reverse the linked list.
c) Concatenation of two linked lists

WAP to implement Stack & Queues using Linked Representation
*/



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

struct node
{
        int data;
        struct node* next;
};

struct  node *rear=NULL, *front =NULL, *top=NULL;

struct node* getnode(int item)
{
        struct node* newn = (struct node*)malloc(sizeof(struct node));
        newn->data = item;
        newn->next = NULL;
        return newn;
}

void display(struct node* head)
{
        if(head == NULL)
        {
```

```
            printf("List is empty.\n");
            return;
        }
        struct node* ptr = head;
        while(ptr)
        {
            printf("%d->", ptr->data);
            ptr = ptr->next;
        }
        printf("\b \b\b \n");
}

struct node* insertfront(struct node* head, int item)
{
        struct node* newn = getnode(item);
        newn->next = head;
        head = newn;
        return head;
}

void swap(int *a, int *b)
{
        int temp;
        temp = *a;
        *a = *b;
        *b = temp;
}


struct node* sort (struct node* head)
{
        int sorted;
        if(head == NULL) return head;
        struct node* ptr = head;
        do
```

```
        {
                ptr = head;
                sorted = 0;
                while(ptr->next)
                {
                        if(ptr->data > ptr->next->data)
                        {
                                swap(&ptr->data, &ptr->next->data);
                                sorted = 1;
                        }
                        ptr = ptr->next;
                }
        } while(sorted == 1);
        return head;
}

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;
}

struct node* concatenate(struct node* head1, struct node* head2)
{
        struct node* ptr = head1;
        while(ptr->next)
        {
```

```
            ptr = ptr->next;
        }
        ptr->next = head2;
        return head1;
}



void qinsert()
{
   struct  node *newnode;
   newnode=(struct node *) malloc(sizeof(struct node));
   printf("Enter the element:\n");
   scanf("%d",&newnode->data);
   newnode->next=NULL;

   if(rear==NULL)
   {
      rear=newnode;
      front=newnode;

   }
   else
   {
      rear->next=newnode;
      rear=newnode;
   }
}

void qdel()
{
   if(front==NULL)
   {
     printf("Queue is empty\n");return;
   }
```

```
    else
    {
        printf("Deleted ele is %d",front->data);
        if(front==rear)
        {
           printf("Queue is empty\n");
           front=NULL; rear=NULL;
        }
        else
        front=front->next;
    }
}

void qdisplay()
{
    struct node *temp;
    if(front ==NULL)
    {
        printf("Queue is empty");
        return;
    }
    temp=front;
    while (temp !=NULL)
    {
        printf("%d ",temp->data);
        temp=temp->next;
    }

}
void spush()
{
    int item;
    struct node *newnode;
    printf("Enter the element\n");
```

```
    scanf("%d",&item);

    newnode=(struct node*)malloc(sizeof(struct node));
    newnode->data=item;
    newnode->next=NULL;
    if(top==NULL)
        top=newnode;
    else
        newnode->next=top;
        top=newnode;
}
void spop()
{
    if(top==NULL)
        printf("stack is empty");
    else
    {

      printf("element removed is %d:", top->data);

      top=top->next;

    }


}

void sdisplay()
{
struct node *temp;
temp=top;
if(top==NULL)
    printf("Stack is empty");
while(temp!=NULL)
{
```

```
    printf("%d",temp->data);
    printf("\n");
    temp=temp->next;
}

}


int main()
{
        printf("Linked list program containing sort, reverse, and concatenate
functions.\n");
        int n1, n2, n, ch, flag = 0;
        int choice;
        struct node* head1 = NULL; struct node* head2 = NULL;
        do
        {
                printf("Enter the choice\n1.Stack\n2.Queue\n3: Linked list 1\n4:
Linked list 2\n5: Exit\n");
                scanf("%d", &n1);
                switch(n1)
                {

          case 1:
             {

                                do
                                { printf("\n1. Push \n2. Display  \n3. Pop\n");
                                printf("\nEnter your choice : ");
                                scanf("%d",&choice);
                                switch(choice)
                                {
                                   case 1: spush(); break;
                                   case 2: sdisplay();break;
                                   case 3: spop(); break;
```

```
                    ;
                }
                }while(choice!=10);




    }


        case 2:
            {
          do
                { printf("\nQueue implementation using linked list\n");
                  printf("\n1. Create \n2. Display  \n3. Delete  \n4. Exit
\n");
                  printf("\nEnter your choice : ");
                  scanf("%d",&choice);
                  switch(choice)
                  { case 1: qinsert(); break;
                    case 2: qdisplay();break;
                    case 3: qdel(); break;

                  }
                  }while(choice!=10);
                }


                case 3:
                      {
                            do
                            {
                                  printf("3: Insert\n4: Sort\n5: Reverse\n6:
Concatenate with list 1\n7: Display list\n8: Go back to main menu\n9:
Exit\n");
```

```
                scanf("%d", &n2);
                switch(n2)
                {
                        case 3: {
                                        printf("Enter item to be
inserted:  ");
                                        scanf("%d", &n);
                                        head1 =
insertfront(head1, n);
                                        break;
                                }
                        case 4: {
                                        head1 = sort(head1);
                                        break;
                                }
                        case 5: {
                                        reverse(&head1);
                                        break;
                                }
                        case 6: {
                                        head1 =
concatenate(head1, head2);
                                        break;
                                }
                        case 7: {
                                        display(head1);
                                        break;
                                }
                        case 8: {
                                        flag = 1;
                                        break;
                                }
                        case 9: {
                                        exit(0);
                                }
```

```
                                        default: printf("Invalid input.\n");
                                }
                                if(flag == 1)
                                {
                                        break;
                                }
                        }while(1);
                        break;
                }
        case 4: {
                        flag = 0;
                        do
                        {
                                printf("3: Insert\n4: Sort\n5: Reverse\n6:
Concatenate with list 1\n7: Display list\n8: Go back to main menu\n9:
Exit\n");
                                scanf("%d", &n2);
                                switch(n2)
                                {
                                        case 3: {
                                                printf("Enter item to be
inserted:  ");
                                                scanf("%d", &n);
                                                head2 =
insertfront(head2, n);
                                                break;
                                            }
                                        case 4: {
                                                head2 = sort(head2);
                                                break;
                                             }
                                        case 5: {
                                                reverse(&head2);
                                                break;
                                             }
```

```
                                        case 6: {
                                                head2 =
concatenate(head2, head1);
                                                break;
                                        }
                                        case 7: {
                                                display(head2);
                                                break;
                                        }
                                        case 8: {
                                                flag = 1;
                                                break;
                                        }
                                        case 9: {
                                                exit(0);
                                        }
                                        default: printf("Invalid input.\n");
                                }
                                if(flag == 1)
                                {
                                        flag = 0; break;
                                }
                        }while(1);
                        break;
                }
        case 9: exit(0);
        default: printf("Invalid input.\n");
        }
    }while(1);
return 0;
}
```