WEEK 7: operations using linked list

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

struct node
{
	int data;
	struct node *next;
};
struct node *head;
struct node *head2;

//stack operations
void push()
{
	struct node *ptr;
	int new_data;
	ptr = (struct node *)malloc(sizeof(struct node));

	if(ptr == NULL)
	{
		printf("\nOVERFLOW!!!");
	}
	else
	{
		printf("\nEnter the Value to be inserted:");
		scanf("%d",&new_data);
```

```
                ptr->data = new_data;
                ptr->next = head;
                head = ptr;
                printf("\nNODE INSERTED AT THE TOP OF THE STACK\n");
        }
}
void pop()
{
        struct node *ptr;
        if(head == NULL)
        {
                printf("EMPTY LIST!!!");
        }
        else
        {
                ptr = head;
                head = ptr->next;
                free(ptr);
                printf("\nNODE DELETED FROM TOP OF THE STACK\n");
        }
}

//queue operations
void enqueue()
{
        struct node *ptr,*temp;
        int new_data;
        ptr = (struct node *)malloc(sizeof(struct node));
```

```
        printf("\nEnter the Value to be inserted:");
        scanf("%d",&new_data);
        ptr->data = new_data;
        if(head == NULL)
        {
                ptr->next = NULL;
                head = ptr;
                printf("\nNODE INSERTED AT REAR OF THE QUEUE\n");
        }
        else
        {
                temp = head;
                while(temp->next != NULL)
                {
                        temp = temp->next;
                }
                temp->next = ptr;
                ptr->next = NULL;
                printf("\nNODE INSERTED AT REAR OF THE QUEUE\n");
        }
}
void dequeue()
{
        struct node *ptr;
        if(head == NULL)
        {
                printf("EMPTY LIST!!!");
        }
        else
```

```
	{
		ptr = head;
		head = ptr->next;
		free(ptr);
		printf("\nNODE DELETED FROM FRONT OF THE QUEUE\n");
	}
}

//Display List
void display()
{
	struct node *ptr;
	ptr = head;
	if(ptr == NULL)
	{
		printf("EMPTY LIST!!!INSERT FEW ELEMENTS!!");
	}
	else
	{
		printf("\n\nLIST-->");
		while(ptr != NULL)
		{
			printf("\t%d",ptr->data);
			ptr = ptr->next;
		}
	}
}

//sort Linked list in ascending order
```

```
void sort()
{
	struct node *ptr = head;
	struct node *temp = NULL;
	int i;

	if(head == NULL)
	{
		return;
	}
	else
	{
		while(ptr != NULL)
		{
			temp = ptr->next;
			while(temp != NULL)
			{
				if(ptr->data >temp->data)
				{
					i = ptr->data;
					ptr->data = temp->data;
					temp->data = i;
				}
				temp = temp->next;
			}
			ptr = ptr->next;
		}
	}
}
```

```
//reverse Linked List
void reverse()
{
	struct node *prev = NULL;
	struct node *next = NULL;
	struct node *ptr = head;
	while(ptr != NULL)
	{
		next = ptr->next;
		ptr->next = prev;
		prev = ptr;
		ptr = next;
	}
	head = prev;
}

//create list
struct node *create_list(struct node *head)
{
	struct node *ptr,*temp;
	int i,n,new_data;

   printf("\nEnter the number of nodes : ");
   scanf("%d",&n);
	head = NULL;
	if(n == 0)
	{
		return head;
```

```
        }
        for(i=1;i<=n;i++)
        {
                ptr = (struct node *)malloc(sizeof(struct node));
                printf("Enter the element to be inserted : ");
                scanf("%d",&new_data);
                ptr->data = new_data;
                if(head == NULL)
                {
                        ptr->next = NULL;
                        head = ptr;
                }
                else
                {
                        temp = head;
                        while(temp->next != NULL)
                        {
                                temp = temp->next;
                        }
                        temp->next = ptr;
                        ptr->next = NULL;
                }
        }
        return head;
}


//concatenate two lists
struct node *concatenate(struct node *head, struct node *head2)
```

```
{
        struct node *ptr;
        if(head == NULL)
        {
                head = head2;
                return head;
        }
        if(head2 == NULL)
        {
                return head;
        }
        ptr = head;
        while(ptr->next != NULL)
        {
                ptr = ptr->next;
        }
        ptr->next = head2;
        return head;
}




int main()
{
        int choice = 0;
        while(1)
        {

                printf("Choose an option from the list:");
```

```
printf("\nSTACK OPERATIONS:\nPUSH\nPOP");
printf("\nQUEUE OPERATIONS:\nENQUEUE\nDEQUEUE");
printf("\n-----------------");
printf("\nDISPLAY\nSORT\nREVERSE\nCONCATENATION\nEXIT\n");
printf("\nEnter your choice:");
scanf("%d",&choice);
switch(choice)
{
        case 1: push();
                        break;
        case 2: pop();
                        break;
        case 3: enqueue();
                        break;
        case 4: dequeue();
                        break;
        case 5: display();
                        break;
        case 6: sort();
                        printf("\nSorted List::");
                        display();
                        break;
        case 7: reverse();
                        printf("\nReversed List::");
                        display();
                        break;
        case 8: printf("\nCreate a Second list-->");
                        head2 = create_list(head2);
                        printf("\nList1:");
```

```
                display();
                struct node *ptr;
                ptr = head2;
                if(ptr == NULL)
                {
                        printf("LIST2 IS EMPTY!!!");
                }
                else
                {
                        printf("\n\nLIST2-->");
                        while(ptr != NULL)
                        {
                                printf("\t%d",ptr->data);
                                ptr = ptr->next;
                        }
                }
                head = concatenate(head,head2);
                printf("\n\nConcatenated List:");
                display();
                break;
        case 9: exit(1);
        default:
                printf("\nINVALID CHOICE!!!\n");

        }
    }
}
```

```
Choose an option from the list:
STACK OPERATIONS:
PUSH
POP
QUEUE OPERATIONS:
ENQUEUE
DEQUEUE
----------------
DISPLAY
SORT
REVERSE
CONCATENATION
EXIT

Enter your choice:1

Enter the Value to be inserted:1

NODE INSERTED AT THE TOP OF THE STACK

Choose an option from the list:
STACK OPERATIONS:
PUSH
POP
QUEUE OPERATIONS:
ENQUEUE
DEQUEUE
----------------
DISPLAY
SORT
REVERSE
CONCATENATION
EXIT
```

```
Enter your choice:1

Enter the Value to be inserted:2

NODE INSERTED AT THE TOP OF THE STACK

Choose an option from the list:
STACK OPERATIONS:
PUSH
POP
QUEUE OPERATIONS:
ENQUEUE
DEQUEUE
-----------------
DISPLAY
SORT
REVERSE
CONCATENATION
EXIT

Enter your choice:1

Enter the Value to be inserted:3

NODE INSERTED AT THE TOP OF THE STACK
```

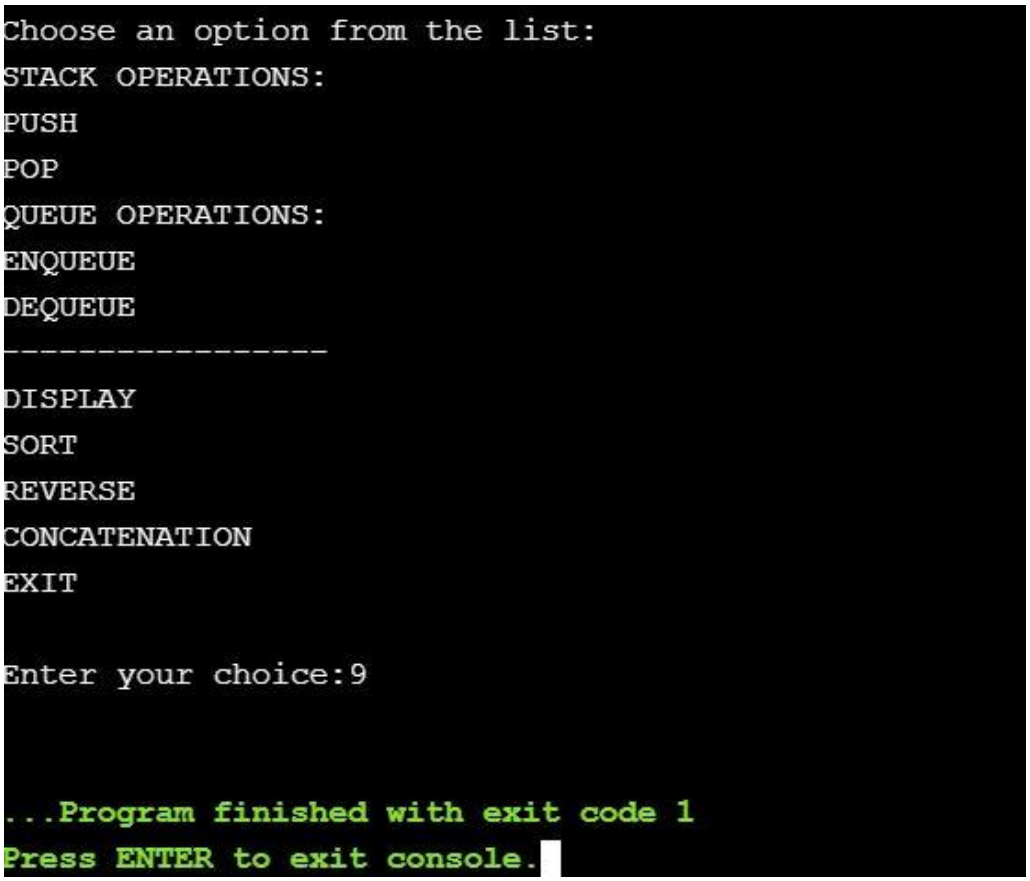
```
Choose an option from the list:
STACK OPERATIONS:
PUSH
POP
QUEUE OPERATIONS:
ENQUEUE
DEQUEUE
----------------
DISPLAY
SORT
REVERSE
CONCATENATION
EXIT

Enter your choice:2

NODE DELETED FROM TOP OF THE STACK

Choose an option from the list:
STACK OPERATIONS:
PUSH
POP
QUEUE OPERATIONS:
ENQUEUE
DEQUEUE
----------------
DISPLAY
SORT
REVERSE
CONCATENATION
EXIT
```

```
Enter your choice:3

Enter the Value to be inserted:45

NODE INSERTED AT REAR OF THE QUEUE

Choose an option from the list:
STACK OPERATIONS:
PUSH
POP
QUEUE OPERATIONS:
ENQUEUE
DEQUEUE
-----------------
DISPLAY
SORT
REVERSE
CONCATENATION
EXIT

Enter your choice:3

Enter the Value to be inserted:67

NODE INSERTED AT REAR OF THE QUEUE
```

```
Choose an option from the list:
STACK OPERATIONS:
PUSH
POP
QUEUE OPERATIONS:
ENQUEUE
DEQUEUE
----------------
DISPLAY
SORT
REVERSE
CONCATENATION
EXIT

Enter your choice:4

NODE DELETED FROM FRONT OF THE QUEUE

Choose an option from the list:
STACK OPERATIONS:
PUSH
POP
QUEUE OPERATIONS:
ENQUEUE
DEQUEUE
----------------
DISPLAY
SORT
REVERSE
CONCATENATION
EXIT
```

```
Enter your choice:5


LIST--> 1       45      67
Choose an option from the list:
STACK OPERATIONS:
PUSH
POP
QUEUE OPERATIONS:
ENQUEUE
DEQUEUE
----------------
DISPLAY
SORT
REVERSE
CONCATENATION
EXIT

Enter your choice:6

Sorted List::

LIST--> 1       45      67
Choose an option from the list:
STACK OPERATIONS:
PUSH
POP
QUEUE OPERATIONS:
ENQUEUE
DEQUEUE
----------------
DISPLAY
SORT
REVERSE
CONCATENATION
EXIT
```

```
Enter your choice:7

Reversed List::

LIST--> 67      45      1
Choose an option from the list:
STACK OPERATIONS:
PUSH
POP
QUEUE OPERATIONS:
ENQUEUE
DEQUEUE
----------------
DISPLAY
SORT
REVERSE
CONCATENATION
EXIT

Enter your choice:8

Create a Second list-->
Enter the number of nodes : 2
Enter the element to be inserted : 12
Enter the element to be inserted : 13

List1:

LIST--> 67      45      1

LIST2-->        12      13

Concatenated List:

LIST--> 67      45      1       12      13
```


Choose an option from the list:
STACK OPERATIONS:
PUSH
POP
QUEUE OPERATIONS:
ENQUEUE
DEQUEUE

DISPLAY
SORT
REVERSE
CONCATENATION
EXIT
Enter your choice:9
...Program finished with exit code 1
Press ENTER to exit console.