WEEK10:

Stacks and Queues using linked list

CODE:

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

struct node
{
	int info;
	struct node *link;
};
typedef struct node *NODE;
NODE getnode()
{
	NODE x;
	x=(NODE)malloc(sizeof(struct node));
	if(x==NULL)
	{
		printf("Memory is full!!\n");
		exit(0);
	}
	return x;
}
void freenode(NODE x)
{
	free(x);
}
NODE insert_front(NODE first,int item)
{
	NODE temp;
	temp=getnode();
	temp->info=item;
	temp->link=NULL;
	if(first==NULL)
		return temp;
	temp->link=first;
	first=temp;
	return first;
```

```
}
NODE insert_rear(NODE first,int item)
{
	NODE temp,cur;
	temp=getnode();
	temp->info=item;
	temp->link=NULL;
	if(first==NULL)
		return temp;
	cur=first;
	while(cur->link!=NULL)
		cur=cur->link;
	cur->link=temp;
	return first;
}
NODE delete_front(NODE first)
{
	NODE temp;
	if(first==NULL)
	{
		printf("Stack is empty cannot pop!\n");
		return first;
	}
	temp=first;
	temp=temp->link;
	printf("The popped item from the stack is : %d\n",first->info);
	free(first);
	return temp;
}
NODE delete_frontq(NODE first)
{
	NODE temp;
	if(first==NULL)
	{
		printf("Queue is empty cannot delete!\n");
		return first;
	}
	temp=first;
	temp=temp->link;
	printf("The deleted item from the queue is : %d\n",first->info);
	free(first);
	return temp;
}
```

```
void display(NODE first)
{
	NODE temp;
	if(first==NULL)
		printf("Stack is EMPTY!\n");
	for(temp=first;temp!=NULL;temp=temp->link)
	{
		printf("%d\n",temp->info);
	}
}
void displayq(NODE first)
{
	NODE temp;
	if(first==NULL)
		printf("Queue is EMPTY!\n");
	for(temp=first;temp!=NULL;temp=temp->link)
	{
		printf("%d\n",temp->info);
	}
}
void main()
{
	int items,itemq,choice;
	NODE firsts=NULL;
	NODE firstq=NULL;

	for(;;)
	{
		printf("\n--------------------------\n1:PUSH into stack\n2:POP from stack\n3:Display
stack\n4:Insert in queue\n5:Delete from queue\n6:Display Queue\n7:Exit\n");
		printf("Enter the choice\n");
		scanf("%d",&choice);
		switch(choice)
		{
			case 1:printf("Enter the item to push into stack\n");
				scanf("%d",&items);
				firsts=insert_front(firsts,items);
				break;
			case 2:firsts=delete_front(firsts);
				break;
			case 3:
				printf("The stack is:\n");
```

```
                display(firsts);
                break;
        case 4:
                printf("Enter the item to be inserted in queue:");
                scanf("%d",&itemq);
                firstq=insert_rear(firstq,itemq);
                break;
        case 5:firstq=delete_frontq(firstq);
                break;

        case 6:
                printf("The Queue is:\n");
                displayq(firstq);
                break;
        case 7: exit(0);break;
        default:printf("INVALID CHOICE!\n");
                break;
    }
  }
}
```

OUTPUT:

```
1:PUSH into stack
2:POP from stack
3:Display stack
4:Insert in queue
5:Delete from queue
6:Display Queue
7:Exit
Enter the choice
1
Enter the item to push into stack
10

-------------------------
1:PUSH into stack
2:POP from stack
3:Display stack
4:Insert in queue
5:Delete from queue
6:Display Queue
7:Exit
Enter the choice
1
Enter the item to push into stack
20

-------------------------
1:PUSH into stack
2:POP from stack
3:Display stack
4:Insert in queue
5:Delete from queue
6:Display Queue
7:Exit
Enter the choice
1
Enter the item to push into stack
30
```

```
1:PUSH into stack
2:POP from stack
3:Display stack
4:Insert in queue
5:Delete from queue
6:Display Queue
7:Exit
Enter the choice
3
The stack is:
30
20
10

--------------------------
1:PUSH into stack
2:POP from stack
3:Display stack
4:Insert in queue
5:Delete from queue
6:Display Queue
7:Exit
Enter the choice
2
The popped item from the stack is : 30

--------------------------
1:PUSH into stack
2:POP from stack
3:Display stack
4:Insert in queue
5:Delete from queue
6:Display Queue
7:Exit
Enter the choice
2
The popped item from the stack is : 20
```

```
1:PUSH into stack
2:POP from stack
3:Display stack
4:Insert in queue
5:Delete from queue
6:Display Queue
7:Exit
Enter the choice
2
The popped item from the stack is : 10

--------------------------
1:PUSH into stack
2:POP from stack
3:Display stack
4:Insert in queue
5:Delete from queue
6:Display Queue
7:Exit
Enter the choice
2
Stack is empty cannot pop!

--------------------------
1:PUSH into stack
2:POP from stack
3:Display stack
4:Insert in queue
5:Delete from queue
6:Display Queue
7:Exit
Enter the choice
4
Enter the item to be inserted in queue:100
```

```
1:PUSH into stack
2:POP from stack
3:Display stack
4:Insert in queue
5:Delete from queue
6:Display Queue
7:Exit
Enter the choice
4
Enter the item to be inserted in queue:200

--------------------------
1:PUSH into stack
2:POP from stack
3:Display stack
4:Insert in queue
5:Delete from queue
6:Display Queue
7:Exit
Enter the choice
4
Enter the item to be inserted in queue:300

--------------------------
1:PUSH into stack
2:POP from stack
3:Display stack
4:Insert in queue
5:Delete from queue
6:Display Queue
7:Exit
Enter the choice
6
The Queue is:
100
200
300
```

```
1:PUSH into stack
2:POP from stack
3:Display stack
4:Insert in queue
5:Delete from queue
6:Display Queue
7:Exit
Enter the choice
5
The deleted item from the queue is : 100

-------------------------
1:PUSH into stack
2:POP from stack
3:Display stack
4:Insert in queue
5:Delete from queue
6:Display Queue
7:Exit
Enter the choice
5
The deleted item from the queue is : 200

-------------------------
1:PUSH into stack
2:POP from stack
3:Display stack
4:Insert in queue
5:Delete from queue
6:Display Queue
7:Exit
Enter the choice
5
The deleted item from the queue is : 300

-------------------------
```

```
-------------------------
1:PUSH into stack
2:POP from stack
3:Display stack
4:Insert in queue
5:Delete from queue
6:Display Queue
7:Exit
Enter the choice
5
Queue is empty cannot delete!
```

Doubly linked list

CODE:

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

struct node
{
	int info;
	struct node *rlink;
	struct node *llink;
};
typedef struct node *NODE;
NODE getnode()
{
	NODE x;
	x = (NODE)malloc(sizeof(struct node));
	if (x == NULL)
	{
		printf("Memory Full!!\n");
		exit(0);
	}
	return x;
}
void freenode(NODE x)
{
	free(x);
}
NODE insert_rear(NODE head, int item)
{
	NODE temp, cur;
	temp = getnode();
	temp->rlink = NULL;
	temp->llink = NULL;
	temp->info = item;
	cur = head->llink;
	temp->llink = cur;
	cur->rlink = temp;
	head->llink = temp;
	temp->rlink = head;
	head->info = head->info + 1;
```

```
        return head;
}
NODE insert_leftpos(int item, NODE head)
{
        NODE temp, cur, prev;
        if (head->rlink == head)
        {
                printf("List is Empty!\n");
                return head;
        }
        cur = head->rlink;
        while (cur != head)
        {
                if (item == cur->info)
                        break;
                cur = cur->rlink;
        }
        if (cur == head)
        {
                printf("Key not found\n");
                return head;
        }
        prev = cur->llink;
        printf("Enter item to be inserted towards left of %d=", item);
        temp = getnode();
        scanf("%d", &temp->info);
        prev->rlink = temp;
        temp->llink = prev;
        cur->llink = temp;
        temp->rlink = cur;
        return head;
}

NODE delete_all_key(int item, NODE head)
{
        NODE prev, cur, next;
        int count;
        if (head->rlink == head)
        {
                printf("List is Empty!");
                return head;
        }
        count = 0;
```

```
	cur = head->rlink;
	while (cur != head)
	{
		if (item != cur->info)
			cur = cur->rlink;
		else
		{
			count++;
			prev = cur->llink;
			next = cur->rlink;
			prev->rlink = next;
			next->llink = prev;
			freenode(cur);
			cur = next;
		}
	}
	if (count == 0)
		printf("Key not found");
	else
		printf("Keys found at % d positions and are deleted\n", count);

	return head;
}


void search_key(int item, NODE head)
{
	NODE prev, cur, next;

	if (head->rlink == head)
	{
		printf("List is Empty!\n");
		return;
	}

	cur = head->rlink;
	while (cur != head)
	{
		if (item != cur->info)
			cur = cur->rlink;
		else
		{
			printf("Item found!\n");return;
```

```
            }

        }
        printf("Item not found!\n");


}



NODE delete_dup(int item, NODE head)
{
        NODE prev, cur, next;
        int count;
        if (head->rlink == head)
        {
                printf("List is Empty!");
                return head;
        }
        count = 0;
        cur = head->rlink;
        while (cur != head)
        {
                if (item != cur->info)
                        cur = cur->rlink;
                else
                {
                        count++;
                        if(count==1)
                        {
                                cur = cur->rlink;
                        }
                        if(count!=1)
                        {
                                prev = cur->llink;
                                next = cur->rlink;
                                prev->rlink = next;
                                next->llink = prev;
                                freenode(cur);
                                cur = next;
                        }

                }
```

```
        }
        if (count == 0)
                printf("Key not found");
        else
                printf("Duplicates are deleted are deleted\n");

        return head;
}

void display(NODE head)
{
        NODE temp;
        if (head->rlink == head)
        {
                printf("List is Empty\n");
                return;
        }
        for (temp = head->rlink; temp != head; temp = temp->rlink)
                printf("%d\n", temp->info);
}
void main()
{
        int item, choice, key;
        NODE head;
        head = getnode();
        head->rlink = head;
        head->llink = head;

        for (;;)
        {
                printf("\n1.Insert_rear\n2.Insert_left_of\n3.Delete_all_key\n4.Delete duplicates\n5.Search
key\n6.Display\n6.Exit\n");
                printf("Enter your choice\n");
                scanf("%d", &choice);
                switch (choice)
                {
                case 1:
                        printf("Enter the item\n");
                        scanf("%d", &item);
                        head = insert_rear(head, item);
                        break;
                case 2:
                        printf("Enter the key item\n");
```

```
                scanf("%d", &item);
                head = insert_leftpos(item, head);
                break;
        case 3:
                printf("Enter the key item to be deleted:\n");
                scanf("%d", &item);
                head = delete_all_key(item, head);
                break;
        case 4:
                printf("Enter the key item whose duplicates are to be deleted:\n");
                scanf("%d", &item);
                head = delete_dup(item, head);
                break;
        case 5:
                printf("Enter the key item to be searched:\n");
                scanf("%d", &item);
                search_key(item, head);
                break;
        case 6:
                printf("The list is:\n");
                display(head);
                break;
        case 7:
                exit(0);
                break;
        default:
                printf("Inavlid choice!\n");
        }
    }
}
```

OUTPUT:

```
1.Insert_rear
2.Insert_left_of
3.Delete_all_key
4.Delete duplicates
5.Search key
6.Display
6.Exit
Enter your choice
1
Enter the item
10

1.Insert_rear
2.Insert_left_of
3.Delete_all_key
4.Delete duplicates
5.Search key
6.Display
6.Exit
Enter your choice
1
Enter the item
20

1.Insert_rear
2.Insert_left_of
3.Delete_all_key
4.Delete duplicates
5.Search key
6.Display
6.Exit
Enter your choice
2
Enter the key item
20
Enter item to be inserted towards left of 20=15
```

```
1.Insert_rear
2.Insert_left_of
3.Delete_all_key
4.Delete duplicates
5.Search key
6.Display
6.Exit
Enter your choice
5
Enter the key item to be searched:
15
Item found!

1.Insert_rear
2.Insert_left_of
3.Delete_all_key
4.Delete duplicates
5.Search key
6.Display
6.Exit
Enter your choice
6
The list is:
10
15
20

1.Insert_rear
2.Insert_left_of
3.Delete_all_key
4.Delete duplicates
5.Search key
6.Display
6.Exit
Enter your choice
1
Enter the item
10
```

```
1.Insert_rear
2.Insert_left_of
3.Delete_all_key
4.Delete duplicates
5.Search key
6.Display
6.Exit
Enter your choice
1
Enter the item
10

1.Insert_rear
2.Insert_left_of
3.Delete_all_key
4.Delete duplicates
5.Search key
6.Display
6.Exit
Enter your choice
6
The list is:
10
15
20
10
10

1.Insert_rear
2.Insert_left_of
3.Delete_all_key
4.Delete duplicates
5.Search key
6.Display
6.Exit
Enter your choice
4
Enter the key item whose duplicates are to be deleted:
10
Duplicates are deleted are deleted
```

```
1.Insert_rear
2.Insert_left_of
3.Delete_all_key
4.Delete duplicates
5.Search key
6.Display
6.Exit
Enter your choice
6
The list is:
10
15
20

1.Insert_rear
2.Insert_left_of
3.Delete_all_key
4.Delete duplicates
5.Search key
6.Display
6.Exit
Enter your choice
3
Enter the key item to be deleted:
20
Keys found at  1 positions and are deleted

1.Insert_rear
2.Insert_left_of
3.Delete_all_key
4.Delete duplicates
5.Search key
6.Display
6.Exit
Enter your choice
6
The list is:
10
15
```