31/10/2020

**Experiment No:12**

**LINKED LIST**

**AIM:**

Write a program to implement operations on Linked List.

**DATA STRUCTURES USED:**

Linked List

**ALGORITHM:**

Algorithm INSERT\_FRONT (ITEM)

1. new = GetNode(Node)
2. If (new = NULL) then
3. Print ”memory underflow”
4. Exit
5. Else
6. new->LINK=HEADER->LINK
7. new-> DATA=ITEM
8. HEADER->LINK=new
9. Endif
10. stop

Algorithm INSERT\_REAR (ITEM)

1. new= GetNodes(Node)
2. if (new = NULL) then
3. print”memory underflow”
4. Exit
5. Else
6. ptr=HEADER
7. While(ptr-> LINK!=NULL)do
8. ptr=ptr->LINK
9. Endwhile
10. ptr->LINK= new
11. new->DATA=ITEM
12. new->LINK=NULL
13. Stop

Algorithm INSERT\_ANY (ITEM,KEY)

1. new= GetNode(Node)
2. if (new = NULL) then
3. print ”memory underflow”
4. Exit
5. Else
6. ptr=HEADER
7. While(ptr->DATA!=KEY) and(ptr-> LINK != NULL)do
8. ptr=ptr->LINK
9. Endwhile
10. If(ptr->LINK=NULL)
11. Print”KEY NOT FOUND”
12. Exit
13. Else
14. new-> LINK=ptr->LINK
15. new->DATA=x
16. ptr->LINK=new
17. Endif
18. Endif
19. stop

Algorithm DELETE\_FRONT

1. ptr=HEADER->LINK
2. if(ptr=NULL)then
3. print “The list is empty”
4. Exit
5. Else
6. HEADER->LINK=ptr->LINK
7. ReturnNode(ptr)
8. stop

Algorithm DELETE\_REAR

1. ptr=HEADER
2. if (ptr->LINK =NULL)then
3. print”the list is empty”
4. exit
5. else
6. while(ptr->LINK!=NULL)
7. ptr1=ptr
8. ptr=ptr->LINK
9. endwhile
10. ptr->LINK=NULL
11. ReturnNode(ptr)
12. Endif
13. Stop

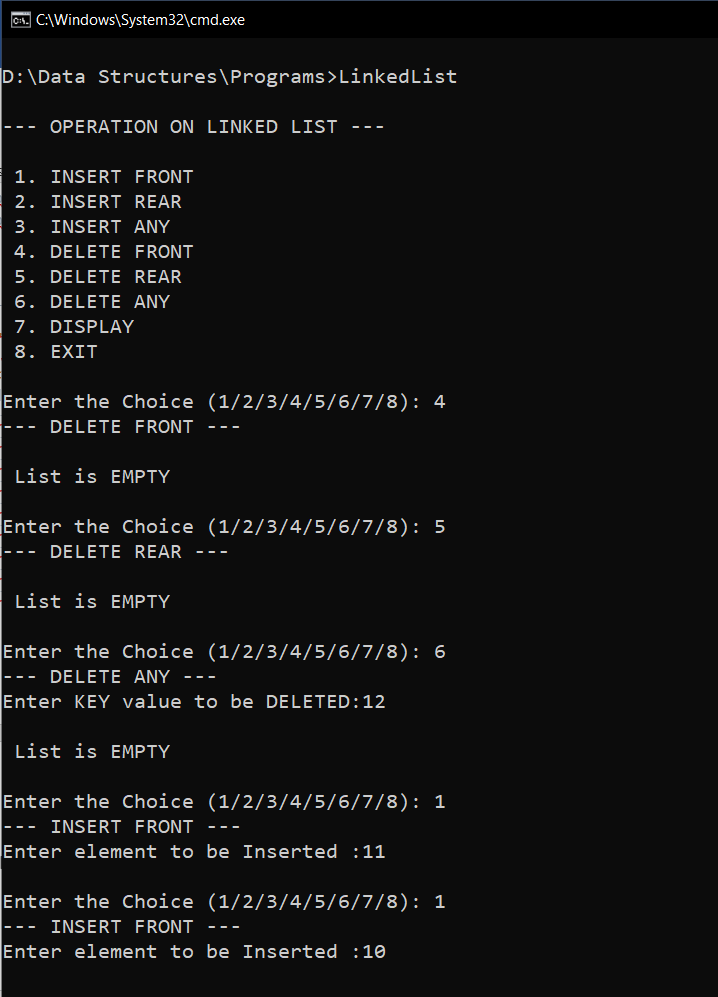
Algorithm DELETE\_ANY

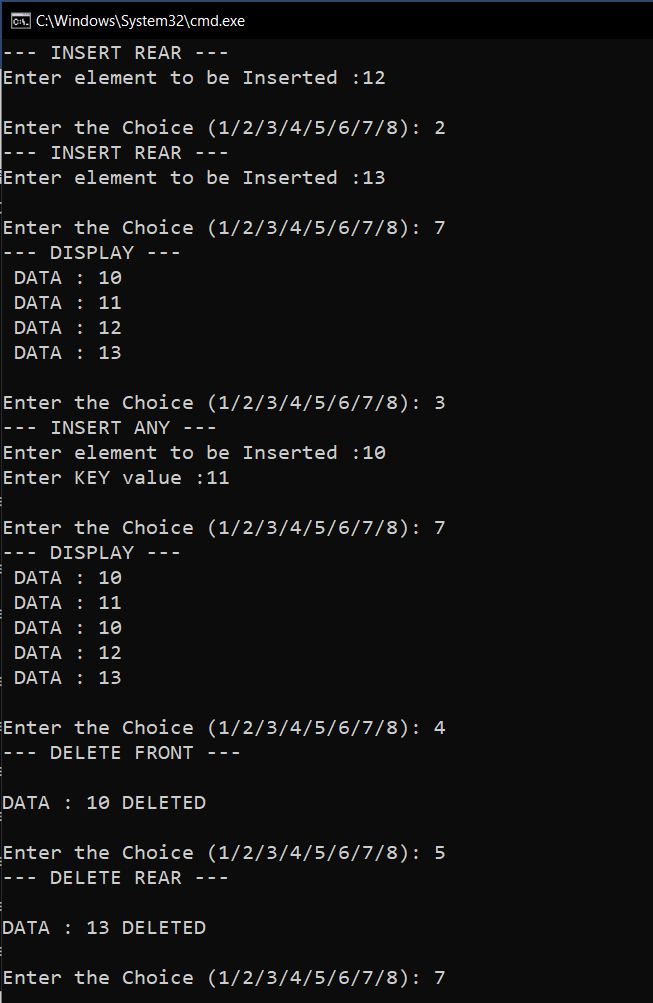
1. ptr1=HEADER
2. ptr=ptr1->LINK
3. while(ptr!=NULL)
4. if(ptr->DATA!= KEY)
5. ptr1=ptr
6. ptr=ptr->LINK
7. else
8. ptr1->LINK=ptr->LINK
9. ReturnNode(ptr)
10. Exit
11. Endif
12. Endwhile
13. If ptr=NULL
14. Print”Node with key doesn’t exist”
15. Endif
16. Stop

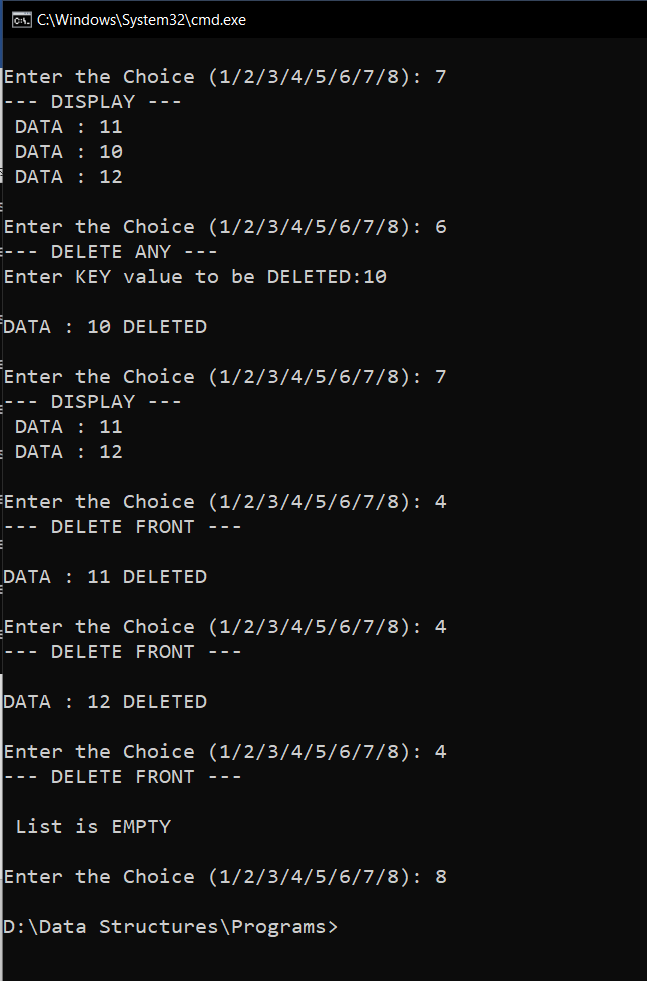
**PROGRAM:**

#include<stdio.h>  
#include<stdlib.h>  
struct node{  
 int data;  
 struct node \*link;  
};  
  
void insert\_front(struct node\* header,int x){  
 struct node\* new = (struct node\*)malloc(sizeof(struct node));  
 new->data=x;  
 new->link=NULL;  
 if(new==NULL){  
 printf("\nMEMORY Underflow\n");  
 }else{  
 if(header->link==NULL){  
 header->link=new;  
 }else{  
 new->link=header->link;  
 header->link=new;  
 }  
 }  
}  
  
void insert\_rear(struct node\* header,int x){  
 struct node\* new = (struct node\*)malloc(sizeof(struct node));  
 new->data=x;  
 new->link=NULL;  
 if(new==NULL){  
 printf("\nMEMORY Underflow\n");  
 }else{  
 if(header->link==NULL){  
 header->link=new;  
 }else{  
 struct node\* ptr=header;  
 while(ptr->link!=NULL){  
 ptr=ptr->link;  
 }  
 ptr->link=new;  
 }  
 }  
}  
void insert\_any(struct node\* header,int x,int key){  
 struct node\* new = (struct node\*)malloc(sizeof(struct node));  
 new->data=x;  
 new->link=NULL;  
 if(new==NULL){  
 printf("\nMEMORY Underflow\n");  
 }else{  
 if(header->link==NULL){  
 printf("KEY Not Found\n");  
 }else{  
 struct node\* ptr=header;  
 while(ptr->data!=key && ptr->link!=NULL){  
 ptr=ptr->link;  
 }  
 if(ptr->data==key){  
 new->link=ptr->link;  
 ptr->link=new;  
 }else{  
 printf("KEY Not Found\n");  
 }  
  
 }  
 }  
}  
void delete\_front(struct node\* header){  
 struct node\* ptr=header->link;  
 if(ptr==NULL){  
 printf("\n List is EMPTY\n");  
 }else{  
 header->link=ptr->link;  
 printf("\nDATA : %d DELETED\n",ptr->data);  
 free(ptr);  
 }  
}  
void delete\_rear(struct node\* header){  
  
 struct node\* ptr=header;  
 struct node\* ptr1;  
 if(ptr->link==NULL){  
 printf("\n List is EMPTY\n");  
 }else{  
 while(ptr->link!=NULL){  
 ptr1=ptr;  
 ptr=ptr->link;  
 }  
 ptr1->link=NULL;  
 printf("\nDATA : %d DELETED\n",ptr->data);  
 free(ptr);  
 }  
}  
void delete\_any(struct node\* header,int key){  
 struct node\* ptr1=header;  
 struct node\* ptr=ptr1->link;  
 if(ptr==NULL){  
 printf("\n List is EMPTY\n");  
 }else{  
 while(ptr->data!=key && ptr->link!=NULL){  
 ptr1=ptr;  
 ptr=ptr->link;  
 }  
 if(ptr->data==key){  
 ptr1->link=ptr->link;  
 printf("\nDATA : %d DELETED\n",ptr->data);  
 free(ptr);  
 }else{  
 printf("\nKEY Not Found\n");  
 }  
 }  
}  
void display(struct node\* header){  
 struct node\* ptr=header;  
 while(ptr->link!=NULL){  
 ptr=ptr->link;  
 printf(" DATA : %d\n",ptr->data);  
 }  
}  
  
void main(){  
 int n,x,y,key;  
 char ans='y';  
 struct node\* header = (struct node\*)malloc(sizeof(struct node));  
 header->link=NULL;  
 printf("\n--- OPERATION ON LINKED LIST --- \n\n");  
 printf(" 1. INSERT FRONT \n");  
 printf(" 2. INSERT REAR \n");  
 printf(" 3. INSERT ANY \n");  
 printf(" 4. DELETE FRONT \n");  
 printf(" 5. DELETE REAR \n");  
 printf(" 6. DELETE ANY \n");  
 printf(" 7. DISPLAY\n");  
 printf(" 8. EXIT\n");  
 while(ans=='y'){  
 printf("\nEnter the Choice (1/2/3/4/5/6/7/8): ");  
 scanf("%d",&n);  
 switch(n){  
 case 1:printf("--- INSERT FRONT ---\n");  
 printf("Enter element to be Inserted :");  
 scanf("%d", &x);  
 insert\_front(header,x);  
 break;  
 case 2:printf("--- INSERT REAR ---\n");  
 printf("Enter element to be Inserted :");  
 scanf("%d", &x);  
 insert\_rear(header,x);  
 break;  
 case 3:printf("--- INSERT ANY ---\n");  
 printf("Enter element to be Inserted :");  
 scanf("%d", &x);  
 printf("Enter KEY value :");  
 scanf("%d", &key);  
 insert\_any(header,x,key);  
 break;  
 case 4:printf("--- DELETE FRONT ---\n");  
 delete\_front(header);  
 break;  
 case 5:printf("--- DELETE REAR ---\n");  
 delete\_rear(header);  
 break;  
 case 6:printf("--- DELETE ANY ---\n");  
 printf("Enter KEY value to be DELETED:");  
 scanf("%d", &key);  
 delete\_any(header,key);  
 break;  
 case 7:printf("--- DISPLAY ---\n");  
 display(header);  
 break;  
 case 8:ans='n';  
 break;  
 default:printf("Enter a Valid Input\n");  
 }  
 }  
}

**OUTPUT:**







**RESULT:**

The given operations are performed on a Linked List.

Time complexity of INSERT\_FRONT() operation is O(1).

Time complexity of INSERT\_REAR() operation is O(n).

Time complexity of INSERT\_ANY() operation is O(n).

Time complexity of DELETE\_FRONT() operation is O(1).

Time complexity of DELETE\_REAR() operation is O(n).

Time complexity of DELETE\_ANY() operation is O(n).