```
 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
1
enter the item at front-end
20

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
1
enter the item at front-end
30

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
3
enter the item at rear-end
40

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
```

```
 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
3
enter the item at rear-end
40

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
3
enter the item at rear-end
70

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
9
30
20
40
70
```

```
 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
7
press 1 for ascending sort and 2 for descending sort:
1

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
9
20
30
40
70

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
8
create a second list
enter the number of elements in second list
5

 press 1 to insert front and 2 to insert rear
1
enter the item at front-end
20
```

```
 press 1 to insert front and 2 to insert rear
1
enter the item at front-end
30

 press 1 to insert front and 2 to insert rear
2
enter the item at rear-end
9

 press 1 to insert front and 2 to insert rear
2
enter the item at rear-end
7

 press 1 to insert front and 2 to insert rear
2
enter the item at rear-end
6

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
9
20
30
40
70
30
20
9
7
6
```

```
iten deleted at rear-end is 6
 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
2
item deleted at front-end is=20

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
6

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
9
7
9
20
30
70
40
30
```

```c
//Queue implementation:-
//Code-
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<process.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("mem full\n");
exit(0);
}
return x;
}
void freenode(NODE x)
{
free(x);
}
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("list is empty cannot delete\n");
return first;
}
temp=first;
temp=temp->link;
printf("item deleted at front-end is=%d\n",first->info);
free(first);
return temp;
}
void display(NODE first)
{
NODE temp;
if(first==NULL)
printf("list empty cannot display items\n");
for(temp=first;temp!=NULL;temp=temp->link)
{
printf("%d\n",temp->info);
}
}
void main()
{
int item,choice,pos;
NODE first=NULL;
for(;;)
{
printf("\n 1:Insert_rear\n 2:Delete_front\n 3:Display_list\n 4:Exit\n");
printf("enter the choice\n");
scanf("%d",&choice);
switch(choice)
{
case 1:printf("enter the item at rear-end\n");
scanf("%d",&item);
first=insert_rear(first,item);
break;
case 2:first=delete_front(first);
break;
case 3:display(first);
break;
default:exit(0);
break;
}
}
```

```
 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
1
enter the item at rear-end
20

 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
1
enter the item at rear-end
40

 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
1
enter the item at rear-end
50

 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
3
20
40
50

 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
2
item deleted at front-end is=20

 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
```

```c
//Stack implementation:-
//Code-
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<process.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("mem 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 delete_front(NODE first)
{
NODE temp;
if(first==NULL)
{
printf("stack is empty cannot delete\n");
```

```c
if(first==NULL)
{
printf("stack is empty cannot delete\n");
return first;
}
temp=first;
temp=temp->link;
printf("item deleted at front-end is=%d\n",first->info);
free(first);
return temp;
}
void display(NODE first)
{
NODE temp;
if(first==NULL)
printf("stack empty cannot display items\n");
for(temp=first;temp!=NULL;temp=temp->link)
{
printf("%d\n",temp->info);
}
}
void main()
{
int item,choice,pos;
NODE first=NULL;
for(;;)
{
printf("\n 1:Insert_front[PUSH]\n 2:Delete_front[pop]\n 3:Display_list\n 4:Exit\n");
printf("enter the choice\n");
scanf("%d",&choice);
switch(choice)
{
case 1:printf("enter the item at front-end\n");
scanf("%d",&item);
first=insert_front(first,item);
break;
case 2:first=delete_front(first);
break;
case 3:display(first);
break;
default:exit(0);
break;
}
}
}
```

```
 1:Insert_front[PUSH]
 2:Delete_front[pop]
 3:Display_list
 4:Exit
enter the choice
1
enter the item at front-end
20

 1:Insert_front[PUSH]
 2:Delete_front[pop]
 3:Display_list
 4:Exit
enter the choice
1
enter the item at front-end
30

 1:Insert_front[PUSH]
 2:Delete_front[pop]
 3:Display_list
 4:Exit
enter the choice
1
enter the item at front-end
70

 1:Insert_front[PUSH]
 2:Delete_front[pop]
 3:Display_list
 4:Exit
enter the choice
2
item deleted at front-end is=70

 1:Insert_front[PUSH]
 2:Delete_front[pop]
 3:Display_list
 4:Exit
enter the choice
3
30
20

 1:Insert_front[PUSH]
 2:Delete_front[pop]
 3:Display_list
 4:Exit
enter the choice
2
item deleted at front-end is=30
```

```c
#include<stdio.h>
#include<stdlib.h>
struct node
{
int info;
struct node *llink;
struct node *rlink;
};
typedef struct node *NODE;
NODE getnode()
{
NODE x;
x=(NODE)malloc(sizeof(struct node));
if(x==NULL)
{
printf("mem full\n");
exit(0);
}
return x;
}
void freenode(NODE x)
{
free(x);
}
NODE dinsert_front(int item,NODE head)
{
NODE temp,cur;
temp=getnode();
temp->info=item;
cur=head->rlink;
head->rlink=temp;
temp->llink=head;
temp->rlink=cur;
cur->llink=temp;
return head;
}
NODE dinsert_rear(int item,NODE head)
{
NODE temp,cur;
temp=getnode();
temp->info=item;
cur=head->llink;
head->llink=temp;
temp->rlink=head;
```

```c
NODE dsearch(int item,NODE head){
NODE temp,cur,prev;
if(head->rlink==head)
{
printf("list empty\n");
return head;
}
cur=head->rlink;
while(cur!=head)
{
if(item==cur->info){
    printf("Key found");
  break;
}
cur=cur->rlink;
}

if(cur==head)
{
 printf("key not found\n");
 return head;
 }
}

NODE ddelete_front(NODE head)
{
NODE cur,next;
if(head->rlink==head)
{
printf("dq empty\n");
return head;
}
cur=head->rlink;
next=cur->rlink;
head->rlink=next;
next->llink=head;
printf("the node deleted is %d",cur->info);
freenode(cur);
return head;
}
NODE ddelete_rear(NODE head)
{
NODE cur,prev;
if(head->rlink==head)
```

```
head->llink=prev;
prev->rlink=head;
printf("the node deleted is %d",cur->info);
freenode(cur);
return head;
}
void display(NODE head)
{
NODE temp;
if(head->rlink==head)
{
printf("dq empty\n");
return;
}
printf("contents of dq\n");
temp=head->rlink;
while(temp!=head)
{
printf("%d\n",temp->info);
temp=temp->rlink;
}
printf("\n");
}

NODE insert_leftpos(int item,NODE head)
{
NODE temp,cur,prev;
if(head->rlink==head)
{
printf("list 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;
 }
```

```
 cur->llink=temp;
 temp->rlink=cur;
 return head;
}
NODE insert_rightpos(int item,NODE head)
{
NODE temp,cur,next;
if(head->rlink==head)
{
printf("list 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;
 }
 next=cur->rlink;
 printf("enter towards right of %d=",item);
 temp=getnode();
 scanf("%d",&temp->info);
 next->llink=temp;
 temp->rlink=next;
 cur->rlink=temp;
 temp->llink=cur;
 return head;
}


int main()
{
NODE head,last;
int item, choice;
head=getnode();
head->rlink=head;
head->llink=head;
for(;;)
{
```

```c
int main()
{
NODE head,last;
int item, choice;
head=getnode();
head->rlink=head;
head->llink=head;
for(;;)
{
printf("\n1:insert front\n2:insert rear\n3:delete front\n4:delete rear\n5:Search\n6:insert left\n7:insert right\n8:displa
printf("enter the choice\n");
scanf("%d",&choice);
switch(choice)
{
case 1: printf("enter the item at front end\n");
scanf("%d",&item);
last=dinsert_front(item,head);
break;
case 2: printf("enter the item at rear end\n");
scanf("%d",&item);
last=dinsert_rear(item,head);
break;
case 3:last=ddelete_front(head);
break;
case 4: last=ddelete_rear(head);
break;
        case 5:printf("enter the key item\n");
scanf("%d",&item);
head=dsearch(item,head);
break;
     case 6:printf("enter the key item\n");
scanf("%d",&item);
head=insert_leftpos(item,head);
break;
    case 7:printf("enter the key item\n");
scanf("%d",&item);
head=insert_rightpos(item,head);
break;
case 8: display(head);
break;
default:exit(0);
}
}
}
```

```
1:insert front
2:insert rear
3:delete front
4:delete rear
5:Search
6:insert left
7:insert right
8:display
9:exit
enter the choice
1
enter the item at front end
20

1:insert front
2:insert rear
3:delete front
4:delete rear
5:Search
6:insert left
7:insert right
8:display
9:exit
enter the choice
2
enter the item at rear end
30

1:insert front
2:insert rear
3:delete front
4:delete rear
5:Search
6:insert left
7:insert right
8:display
9:exit
enter the choice
1
enter the item at front end
60

1:insert front
2:insert rear
3:delete front
4:delete rear
5:Search
6:insert left
7:insert right
8:display
9:exit
enter the choice
8
contents of dq
60
20
30
```

```
1:insert front
2:insert rear
3:delete front
4:delete rear
5:Search
6:insert left
7:insert right
8:display
9:exit
enter the choice
5
enter the key item
20
Key found
1:insert front
2:insert rear
3:delete front
4:delete rear
5:Search
6:insert left
7:insert right
8:display
9:exit
enter the choice
4
the node deleted is 60
1:insert front
2:insert rear
3:delete front
4:delete rear
5:Search
6:insert left
7:insert right
8:display
9:exit
```

```
4:delete rear
5:Search
6:insert left
7:insert right
8:display
9:exit
enter the choice
5
enter the key item
20
Key found
1:insert front
2:insert rear
3:delete front
4:delete rear
5:Search
6:insert left
7:insert right
8:display
9:exit
enter the choice
4
the node deleted is 60
1:insert front
2:insert rear
3:delete front
4:delete rear
5:Search
6:insert left
7:insert right
8:display
9:exit
```

```c
#include<conio.h>
#include<process.h>
struct node
{
int info;
struct node*llink;
struct node*rlink;
};
typedef struct node*NODE;
NODE getnode()
{
NODE x;
x=(NODE)malloc(sizeof(struct node));
if(x==NULL)
{
printf("memory not available");
exit(0);
}
return x;
}
void freenode(NODE x)
{
free(x);
}
NODE insert(int item,NODE root)
{
NODE temp,cur,prev;
char direction[10];
int i;
temp=getnode();
temp->info=item;
temp->llink=NULL;
temp->rlink=NULL;
if(root==NULL)
 return temp;
printf("give direction to insert\n");
scanf("%s",direction);
prev=NULL;
cur=root;
for(i=0;i<strlen(direction)&&cur!=NULL;i++)
{
prev=cur;
if(direction[i]=='l')
cur=cur->llink;
else
```

```
50  printf("insertion not possible\n");
51  freenode(temp);
52  return(root);
53  }
54  if(cur==NULL)
55  {
56  if(direction[i-1]=='l')
57  prev->llink=temp;
58  else
59  prev->rlink=temp;
60  }
61  return(root);
62  }
63  void preorder(NODE root)
64  {
65  if(root!=NULL)
66  {
67  printf("the item is %d\n",root->info);
68  preorder(root->llink);
69  preorder(root->rlink);
70  }
71  }
72  void inorder(NODE root)
73  {
74  if(root!=NULL)
75  {
76  inorder(root->llink);
77  printf("the item is%d\n",root->info);
78  inorder(root->rlink);
79  }
80  }
81  void postorder(NODE root)
82  {
83  if (root!=NULL)
84  {
85  postorder(root->llink);
86  postorder(root->rlink);
87  printf("the item is%d\n",root->info);
88  }
89  }
90  void display(NODE root,int i)
91  {
92  int j;
93  if(root!=NULL)
94  {
```

```
{
display(root->rlink,i+1);
for (j=1;j<=i;j++)
printf("   ");
printf("%d\n",root->info);
display(root->llink,i+1);
}
}

void main()
{
NODE root=NULL;
int choice,i,item;

for(;;)
{
printf("1.insert\n2.preorder\n3.inorder\n4.postorder\n5.display\n");
printf("enter the choice\n");
scanf("%d",&choice);
switch(choice)
{
case 1: printf("enter the item\n");
scanf("%d",&item);
root=insert(item,root);
break;
case 2: if(root==NULL)
{
printf("tree is empty");
}
else
{
printf("given tree is");
display(root,1);
printf("the preorder traversal is \n");
preorder(root);
}
break;
case 3:if(root==NULL)
  {
printf("tree is empty");
  }
  else
  {
printf("given tree is");
```

```
root=insert(item,root);
break;
case 2: if(root==NULL)
{
printf("tree is empty");
}
else
{
printf("given tree is");
display(root,1);
printf("the preorder traversal is \n");
preorder(root);
}
break;
case 3:if(root==NULL)
  {
printf("tree is empty");
  }
  else
  {
printf("given tree is");
display(root,1);
printf("the inorder traversal is \n");
inorder(root);
}
  break;
case 4:if (root==NULL)
{
printf("tree is empty");
}
  else
  {
printf("given tree is");
display(root,1);
printf("the postorder traversal is \n");
postorder(root);
  }
 break;
case 5:display(root,1);
  break;
default:exit(0);
}
}
}
```

```
1.insert
2.preorder
3.inorder
4.postorder
5.display
enter the choice
1
enter the item
20
1.insert
2.preorder
3.inorder
4.postorder
5.display
enter the choice
1
enter the item
30
give direction to insert
3
1.insert
2.preorder
3.inorder
4.postorder
5.display
enter the choice
1
enter the item
40
```

```
1.insert
2.preorder
3.inorder
4.postorder
5.display
enter the choice
2
given tree is    30
  20
the preorder traversal is
the item is 20
the item is 30
1.insert
2.preorder
3.inorder
4.postorder
5.display
enter the choice
4
given tree is    30
  20
the postorder traversal is
the item is30
the item is20
1.insert
2.preorder
3.inorder
4.postorder
5.display
enter the choice
5
    30
  20
1.insert
2.preorder
3.inorder
4.postorder
5.display
enter the choice
3
given tree is    30
  20
the inorder traversal is
the item is20
the item is30
1.insert
2.preorder
3.inorder
4.postorder
5.display
enter the choice
-
```