

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

```
9
20
30
40
70
30
20
9
7
6
```



item 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:concat  
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:concat  
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:concat  
9:display\_list  
10:Exit

enter the choice

9

7

9

20

30

70

40

30

```

1 //Queue implementation:-
2 //code-
3 #include<stdio.h>
4 #include<conio.h>
5 #include<stdlib.h>
6 #include<process.h>
7 struct node
8 {
9     int info;
10    struct node *link;
11 };
12 typedef struct node *NODE;
13 NODE getnode()
14 {
15     NODE x;
16     x=(NODE)malloc(sizeof(struct node));
17     if(x==NULL)
18     {
19         printf("mem full\n");
20         exit(0);
21     }
22     return x;
23 }
24 void freenode(NODE x)
25 {
26     free(x);
27 }
28 NODE insert_rear(NODE first,int item)
29 {
30     NODE temp,cur;
31     temp=getnode();
32     temp->info=item;
33     temp->link=NULL;
34     if(first==NULL)
35         return temp;
36     cur=first;
37     while(cur->link!=NULL)
38         cur=cur->link;
39     cur->link=temp;
40     return first;
41 }
42
43 NODE delete_front(NODE first)
44 {
45     NODE temp;

```



```

46 if(first==NULL)
47 {
48     printf("list is empty cannot delete\n");
49     return first;
50 }
51 temp=first;
52 temp=temp->link;
53 printf("item deleted at front-end is=%d\n",first->info);
54 free(first);
55 return temp;
56 }
57 void display(NODE first)
58 {
59     NODE temp;
60     if(first==NULL)
61         printf("list empty cannot display items\n");
62     for(temp=first;temp!=NULL;temp=temp->link)
63     {
64         printf("%d\n",temp->info);
65     }
66 }
67 void main()
68 {
69     int item,choice,pos;
70     NODE first=NULL;
71     for(;;)
72     {
73         printf("\n 1:Insert_rear\n 2:Delete_front\n 3:Display_list\n 4:Exit\n");
74         printf("enter the choice\n");
75         scanf("%d",&choice);
76         switch(choice)
77         {
78             case 1:printf("enter the item at rear-end\n");
79                     scanf("%d",&item);
80                     first=insert_rear(first,item);
81                     break;
82             case 2:first=delete_front(first);
83                     break;
84             case 3:display(first);
85                     break;
86             default:exit(0);
87             break;
88         }
89     }

```

```
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
```



```

1 //Stack implementation:-
2 //Code-
3 #include<stdio.h>
4 #include<conio.h>
5 #include<stdlib.h>
6 #include<process.h>
7 struct node
8 {
9     int info;
10    struct node *link;
11 };
12 typedef struct node *NODE;
13 NODE getnode()
14 {
15     NODE x;
16     x=(NODE)malloc(sizeof(struct node));
17     if(x==NULL)
18     {
19         printf("mem full\n");
20         exit(0);
21     }
22     return x;
23 }
24 void freenode(NODE x)
25 {
26     free(x);
27 }
28 NODE insert_front(NODE first,int item)
29 {
30     NODE temp;
31     temp=getnode();
32     temp->info=item;
33     temp->link=NULL;
34     if(first==NULL)
35         return temp;
36     temp->link=first;
37     first=temp;
38     return first;
39 }
40 NODE delete_front(NODE first)
41 {
42     NODE temp;
43     if(first==NULL)
44     {
45         printf("stack is empty cannot delete\n");

```

```

43 if(first==NULL)
44 {
45     printf("stack is empty cannot delete\n");
46     return first;
47 }
48 temp=first;
49 temp=temp->link;
50 printf("item deleted at front-end is=%d\n",first->info);
51 free(first);
52 return temp;
53 }
54 void display(NODE first)
55 {
56     NODE temp;
57     if(first==NULL)
58         printf("stack empty cannot display items\n");
59     for(temp=first;temp!=NULL;temp=temp->link)
60     {
61         printf("%d\n",temp->info);
62     }
63 }
64 void main()
65 {
66     int item,choice,pos;
67     NODE first=NULL;
68     for(;;)
69     {
70         printf("\n 1:Insert_front[PUSH]\n 2:Delete_front[pop]\n 3:Display_list\n 4:Exit\n");
71         printf("enter the choice\n");
72         scanf("%d",&choice);
73         switch(choice)
74         {
75             case 1:printf("enter the item at front-end\n");
76                     scanf("%d",&item);
77                     first=insert_front(first,item);
78                     break;
79             case 2:first=delete_front(first);
80                     break;
81             case 3:display(first);
82                     break;
83             default:exit(0);
84             break;
85         }
86     }
87 }

```



```
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
```



```
1  #include<stdio.h>
2  #include<stdlib.h>
3  struct node
4  {
5      int info;
6      struct node *llink;
7      struct node *rlink;
8  };
9  typedef struct node *NODE;
10 NODE getnode()
11 {
12     NODE x;
13     x=(NODE)malloc(sizeof(struct node));
14     if(x==NULL)
15     {
16         printf("mem full\n");
17         exit(0);
18     }
19     return x;
20 }
21 void freenode(NODE x)
22 {
23     free(x);
24 }
25 NODE dinser_front(int item,NODE head)
26 {
27     NODE temp,cur;
28     temp=getnode();
29     temp->info=item;
30     cur=head->rlink;
31     head->rlink=temp;
32     temp->llink=head;
33     temp->rlink=cur;
34     cur->llink=temp;
35     return head;
36 }
37 NODE dinser_rear(int item,NODE head)
38 {
39     NODE temp,cur;
40     temp=getnode();
41     temp->info=item;
42     cur=head->llink;
43     head->llink=temp;
44     temp->rlink=cur;
```

```
50 NODE dsearch(int item,NODE head){
51     NODE temp,cur,prev;
52     if(head->rlink==head)
53     {
54         printf("list empty\n");
55         return head;
56     }
57     cur=head->rlink;
58     while(cur!=head)
59     {
60         if(item==cur->info){
61             printf("Key found");
62             break;
63         }
64         cur=cur->rlink;
65     }
66     if(cur==head)
67     {
68         printf("key not found\n");
69         return head;
70     }
71 }
72
73
74 NODE ddelete_front(NODE head)
75 {
76     NODE cur,next;
77     if(head->rlink==head)
78     {
79         printf("dq empty\n");
80         return head;
81     }
82     cur=head->rlink;
83     next=cur->rlink;
84     head->rlink=next;
85     next->llink=head;
86     printf("the node deleted is %d",cur->info);
87     freenode(cur);
88     return head;
89 }
90 NODE ddelete_rear(NODE head)
91 {
92     NODE cur,prev;
93     if(head->rlink==head)
```



```
100 head->llink=prev;
101 prev->rlink=head;
102 printf("the node deleted is %d",cur->info);
103 freenode(cur);
104 return head;
105 }
106 void display(NODE head)
107 {
108     NODE temp;
109     if(head->rlink==head)
110     {
111         printf("dq empty\n");
112         return;
113     }
114     printf("contents of dq\n");
115     temp=head->rlink;
116     while(temp!=head)
117     {
118         printf("%d\n",temp->info);
119         temp=temp->rlink;
120     }
121     printf("\n");
122 }
123
124 NODE insert_leftpos(int item,NODE head)
125 {
126     NODE temp,cur,prev;
127     if(head->rlink==head)
128     {
129         printf("list empty\n");
130         return head;
131     }
132     cur=head->rlink;
133     while(cur!=head)
134     {
135         if(item==cur->info)break;
136         cur=cur->rlink;
137     }
138     if(cur==head)
139     {
140         printf("key not found\n");
141         return head;
142     }
```



```

149     cur->llink=temp;
150     temp->rlink=cur;
151     return head;
152 }
153 NODE insert_rightpos(int item,NODE head)
154 {
155     NODE temp,cur,next;
156     if(head->rlink==head)
157     {
158         printf("list empty\n");
159         return head;
160     }
161     cur=head->rlink;
162     while(cur!=head)
163     {
164         if(item==cur->info)break;
165         cur=cur->rlink;
166     }
167     if(cur==head)
168     {
169         printf("key not found\n");
170         return head;
171     }
172     next=cur->rlink;
173     printf("enter towards right of %d=",item);
174     temp=getnode();
175     scanf("%d",&temp->info);
176     next->llink=temp;
177     temp->rlink=next;
178     cur->rlink=temp;
179     temp->llink=cur;
180     return head;
181 }
182
183
184 int main()
185 {
186     NODE head,last;
187     int item, choice;
188     head=getnode();
189     head->rlink=head;
190     head->llink=head;
191     for(;;)
192     {

```

```

184 int main()
185 {
186     NODE head, last;
187     int item, choice;
188     head=getnode();
189     head->rlink=head;
190     head->llink=head;
191     for(;;)
192     {
193         printf("\n1:insert front\n2:insert rear\n3:delete front\n4:delete rear\n5:Search\n6:insert left\n7:insert right\n8:display\n");
194         printf("enter the choice\n");
195         scanf("%d",&choice);
196         switch(choice)
197         {
198             case 1: printf("enter the item at front end\n");
199                     scanf("%d",&item);
200                     last=dinsert_front(item,head);
201                     break;
202             case 2: printf("enter the item at rear end\n");
203                     scanf("%d",&item);
204                     last=dinsert_rear(item,head);
205                     break;
206             case 3: last=ddelete_front(head);
207                     break;
208             case 4: last=ddelete_rear(head);
209                     break;
210             case 5: printf("enter the key item\n");
211                     scanf("%d",&item);
212                     head=dsearch(item,head);
213                     break;
214             case 6: printf("enter the key item\n");
215                     scanf("%d",&item);
216                     head=insert_leftpos(item,head);
217                     break;
218             case 7: printf("enter the key item\n");
219                     scanf("%d",&item);
220                     head=insert_rightpos(item,head);
221                     break;
222             case 8: display(head);
223                     break;
224             default: exit(0);
225         }
226     }
227 }

```



```
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
```

```

1 #include<conio.h>
2 #include<process.h>
3 struct node
4 {
5     int info;
6     struct node*llink;
7     struct node*rlink;
8 };
9 typedef struct node*NODE;
10 NODE getnode()
11 {
12     NODE x;
13     x=(NODE)malloc(sizeof(struct node));
14     if(x==NULL)
15     {
16         printf("memory not available");
17         exit(0);
18     }
19     return x;
20 }
21 void freenode(NODE x)
22 {
23     free(x);
24 }
25 NODE insert(int item,NODE root)
26 {
27     NODE temp,cur,prev;
28     char direction[10];
29     int i;
30     temp=getnode();
31     temp->info=item;
32     temp->llink=NULL;
33     temp->rlink=NULL;
34     if(root==NULL)
35         return temp;
36     printf("give direction to insert\n");
37     scanf("%s",direction);
38     prev=NULL;
39     cur=root;
40     for(i=0;i<strlen(direction)&&cur!=NULL;i++)
41     {
42         prev=cur;
43         if(direction[i]=='l')
44             cur=cur->llink;
45         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 {

```



```
117 root=insert(item,root);
118 break;
119 case 2: if(root==NULL)
120 {
121 printf("tree is empty");
122 }
123 else
124 {
125 printf("given tree is");
126 display(root,1);
127 printf("the preorder traversal is \n");
128 preorder(root);
129 }
130 break;
131 case 3:if(root==NULL)
132 {
133 printf("tree is empty");
134 }
135 else
136 {
137 printf("given tree is");
138 display(root,1);
139 printf("the inorder traversal is \n");
140 inorder(root);
141 }
142 break;
143 case 4:if (root==NULL)
144 {
145 printf("tree is empty");
146 }
147 else
148 {
149 printf("given tree is");
150 display(root,1);
151 printf("the postorder traversal is \n");
152 postorder(root);
153 }
154 break;
155 case 5:display(root,1);
156 break;
157 default:exit(0);
158 }
159 }
160 }
161
```



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