

Lab 7 Program : 25/11/20

IBM19CS01  
RAHIL

WAP to Implement LL with following operations.

- a) Sorting    b) Reversing    c) Concatenation.

code:

```
#include <stdio.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 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("List is Empty . Cannot Delete!! \n");

return first;

}

temp = first;

temp = temp -> link;

printf("Item Deleted at front-end = %d \n", first -> info);

free (first);

return temp;

}

IBM1923070

RAHIL

NODE delete → rear (NODE first)

```
{
    NODE cur, prev;
    if (first == NULL)
    {
        printf ("List is Empty . Cannot Delete !! \n");
        return first;
    }
    if (first → link == NULL)
    {
        printf ("Item deleted at Rear - end = %.d \n", first → info);
        free (first);
        return NULL;
    }
    prev = NULL;
    cur = first;
    while (cur → link != NULL)
    {
        prev = cur;
        cur = cur → link;
    }
    printf ("Item Deleted at Rear - end = %.d \n", cur → info);
    free (cur);
    prev → link = NULL;
    return first;
}

void display (NODE first)
{
    NODE temp;
    if (first == NULL)
        printf ("Empty List . Cannot Display Items . \n");
    for (temp = first; temp != NULL; temp = temp → link)
        printf ("%.d \n", temp → info);
}
```

NODE sort (NODE first)

```
{  
    NODE temp1;  
    NODE temp2;  
    for (temp1 = first; temp1 != NULL; temp1 = temp1 -> link)  
    {  
        for (temp2 = temp1 -> link; temp2 != NULL; temp2 = temp2 -> link)  
        {  
            first = temp1 -> info;  
            temp1 -> info = temp2 -> info;  
            temp2 -> info = first;  
        }  
    }  
}
```

NODE reverse (NODE first)

```
{  
    NODE cur, temp;  
    cur = NULL;  
    while (first != NULL)  
    {  
        temp = first;  
        first = first -> link;  
        temp -> link = cur;  
        cur = temp;  
    }  
    return cur;  
}
```

NODE concat (NODE first, NODE sec)

```
{  
    NODE cur;  
    if (first == NULL) return sec;  
    cur = first -> link;  
    while (cur != NULL)  
    {  
        cur = cur -> link;  
    }  
    cur = sec;  
}
```

```
if (sec == NULL)
    return first;
```

```
cur = first;
```

```
while (cur -> link != NULL)
```

```
    cur = cur -> link;
```

```
cur -> link = sec;
```

```
return first;
```

```
}
```

```
void main()
```

```
{
```

```
    int item, choice, key, pos, n;
```

```
    int count = 0;
```

```
    NODE first = NULL;
```

```
    for (;;) 
```

```
    {
```

```
        printf("\n 1. Insert-front\n 2. Insert-Rear\n 3. Delete-Front\n 4. Delete-Rear\n 5. Display List\n 6. Sort List\n 7. Reverse\n 8. Concatenate\n 9. Exit");
```

```
        printf("Enter choice: ");
```

```
        scanf("%d", &choice);
```

```
        switch (choice)
```

```
        {
```

```
            case 1:
```

```
                printf("Enter the item at front-end: ");
```

```
                scanf("%d", &item);
```

```
                first = insert-front (first, item);
```

```
                break;
```

```
            case 2:
```

```
                printf("Enter the item at Rear-end: ");
```

```
                scanf("%d", &item);
```

```
                first = insert-rear (first, item);
```

```
                break;
```



case 3:

first = delete-front(first);

break;

case 4:

first = delete-rear(first);

break;

case 5:

display(first);

break;

case 6:

Sort(first);

display(first);

break;

case 7:

first = reverse(first);

display(first);

break;

case 8:

printf("Enter No. of Nodes in List 1: ");

scanf("%d", &n);

NODE a = NULL;

for (int i = 0; i < n; i++)

{  
printf("Enter Item: ");

scanf("%d", &item);

a = insert-rear(a, item);

}

printf("Enter No. of Nodes in List 2: ");

scanf("%d", &n);

```
Node b = NULL;
```

```
for (int i = 0; i < n; i++)
```

```
{
```

```
    printf ("Enter Item: ");
```

```
    scanf ("%d", &item);
```

```
    b = insert_rear (b, item);
```

```
}
```

```
a = concat (a, b);
```

```
display (a);
```

```
break;
```

```
default:
```

```
    exit (0);
```

```
    break;
```

```
}
```

```
}
```

```
}
```

```

Saved
SortRevCon.c
#include<stdio.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 full\n");
        exit(1);
    }
    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("List is Empty. Cannot Delete!\n");
        return first;
    }
    temp=first;
    temp=temp->link;
    printf("Item Deleted at Front-end = %d\n",first->info);
    free(first);
    return temp;
}
NODE delete_rear(NODE first)
{
    NODE cur,prev;
    if(first==NULL)
    {
        printf("List is Empty. Cannot Delete!\n");
        return first;
    }
    if(first->link==NULL)
    {
        printf("Item Deleted at Rear-end = %d\n",first->info);
        free(first);
        return NULL;
    }
    prev=NULL;
    cur=first;
    while(cur->link!=NULL)
    {
        prev=cur;
        cur=cur->link;
    }
    printf("Item Deleted at Rear-end = %d\n",cur->info);
    free(cur);
    prev->link=NULL;
    return first;
}
void display(NODE first)
{
    NODE temp;
    if(first==NULL)
    {
        printf("Empty List. Cannot Display Items.\n");
    }
    for(temp=first;temp!=NULL;temp=temp->link)
    {
        printf("%d\n",temp->info);
    }
}
NODE sort(NODE first)
{
    NODE temp1;
    NODE temp2;
    for(temp1=first;temp1!=NULL;temp1=temp1->link)
    {
        for(temp2=temp1->link;temp2!=NULL;temp2=temp2->link)
        {
            if(temp1->info>temp2->info)
            {
                temp1->info=temp2->info;
                temp2->info=temp1->info;
            }
        }
    }
}
NODE reverse(NODE first)
{
    NODE cur,temp;
    cur=NULL;
    while(first!=NULL)
    {
        temp=first;
        first=first->link;
        temp->link=cur;
        cur=temp;
    }
    return cur;
}
NODE concat(NODE first,NODE sec)
{
    NODE cur;
    if(first==NULL)
    {
        return sec;
    }
    if(sec==NULL)
    {
        return first;
    }
    cur=first;
    while(cur->link!=NULL)
    {
        cur=cur->link;
    }
    cur->link=sec;
    return first;
}
void main()
{
    int item,choice,key,pos,n;
    int count=0;
    NODE first=NULL;
    int i;
    printf("\n Insert_Front\n2.Insert_Rear\n3.Delete_Front\n4.Delete_Rear\n");
    printf("Enter Choice:");
    scanf("%d",&choice);
    switch(choice)
    {
        case 1:
            printf("Enter the item at Front-end: ");
            scanf("%d",&item);
            first=insert_front(first,item);
            break;
        case 2:
            printf("Enter the item at Rear-end:");
            scanf("%d",&item);
            first=insert_rear(first,item);
            break;
        case 3:
            first=delete_front(first);
            break;
        case 4:
            first=delete_rear(first);
            break;
        case 5:
            display(first);
            break;
        case 6:
            first=sort(first);
            display(first);
            break;
        case 7:
            first=reverse(first);
            display(first);
            break;
        case 8:
            printf("Enter No. of Nodes in List 1 :");
            scanf("%d",&n);
            NODE a=NULL;
            for(int i=0;i<n;i++)
            {
                printf("Enter item:");
                scanf("%d",&item);
                a=insert_rear(a,item);
            }
            printf("Enter No. of Nodes in List 2 :");
            scanf("%d",&n);
            NODE b=NULL;
            for(int i=0;i<n;i++)
            {
                printf("Enter item:");
                scanf("%d",&item);
                b=insert_rear(b,item);
            }
            a=concat(a,b);
            display(a);
            break;
        default:
            exit(1);
            break;
    }
}
}

```



```

x Terminal
1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Display List
6.Sort List
7.Reverse
8.Concatenate Two Lists
9.Exit
Enter Choice:1
Enter the Item at Front-end: 30

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Display List
6.Sort List
7.Reverse
8.Concatenate Two Lists
9.Exit
Enter Choice:1
Enter the Item at Front-end: 31

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Display List
6.Sort List
7.Reverse
8.Concatenate Two Lists
9.Exit
Enter Choice:1
Enter the Item at Front-end: 32

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Display List
6.Sort List
7.Reverse
8.Concatenate Two Lists
9.Exit
Enter Choice:5
32
31
30

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Display List
6.Sort List
7.Reverse
8.Concatenate Two Lists
9.Exit
Enter Choice:6
30
31
32

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Display List
6.Sort List
7.Reverse
8.Concatenate Two Lists
9.Exit
Enter Choice:7
32
31
30

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Display List
6.Sort List
7.Reverse
8.Concatenate Two Lists
9.Exit
Enter Choice:8
Enter No. of Nodes in List 1 :2
Enter Item:50
Enter Item:60
Enter No. of Nodes in List 2 :3
Enter Item:70
Enter Item:80
Enter Item:90
50
60
70
80
90

```

code:

```
#include <stdio.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_rear (NODE first)
{
    NODE cur, prev;
    if (first == NULL)
    {
        printf ("List is Empty Cannot Delete \n");
        return first;
    }
    if (first->link == NULL)
    {
        printf ("Item deleted is %d \n", first->info);
        free (first);
        return NULL;
    }
    prev = NULL;
    cur = first;
    while (cur->link != NULL)
    {
        prev = cur;
        cur = cur->link;
    }
    printf ("Item Deleted at rear-end is %d", cur->info);
    free (cur);
    prev->link = NULL;
```

return first;

}  
NODE insert\_pos (int item, int pos, NODE first)

{

NODE temp, cur, prev;

int count;

temp = getnode ();

temp -> info = item;

temp -> link = NULL;

if (first == NULL && pos == 1)

{  
return temp;

}

if (first == NULL)

{  
printf ("Invalid ~~Exp~~ Position\n");  
return first;

}

if (pos == 1)

{  
temp -> link = first;  
first = temp;  
return temp;

}

count = 1;

prev = NULL;

cur = first;

while (cur != NULL && count != pos)

{

prev = cur;

cur = cur -> link;

count ++;

}

```

if (count == pos)
{
    prev → link = temp;
    temp → link = cur;
    return first;
}
printf ("Invalid position\n");
return first;
}
NODE delete_pos (int pos, NODE first)
{
    NODE cur;
    NODE prev;
    int count, flag = 0;
    if (first == NULL || pos < 0)
    {
        printf ("Invalid position\n");
        return NULL;
    }
    if (pos == 1)
    {
        cur = first;
        first = first → link;
        free node (cur);
        return first;
    }
    prev = NULL;
    cur = first;
    count = 1;
    while (cur != NULL)
    {
        if (count == pos) { flag = 1; break; }
        count ++;
        prev = cur;
    }

```

```

    cur = cur -> link;
}
if (flag == 0)
{
    printf("Invalid position \n");
    return first;
}

printf("Item deleted at given position is %d \n", cur -> info);
prev -> link = cur -> link;
freeNode(cur);
return first;
}

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, key, pos;
    int count = 0;
    NODE first = NULL;
    h();
    {
        printf("\n 1. Insert - rear \n 2. Delete - rear \n 3. Insert - info - position \n 4. Delete - info - position \n 5. Display - list \n 6. Exit \n");
        printf("Enter choice: \n");
    }
}

```



```
scanf ("%d", &choice);
```

```
switch (choice);
```

```
{
```

```
case 1:
```

```
printf ("Enter item at rear-end\n");
```

```
scanf ("%d", &itemchoice);
```

```
first = insert_rear (first, item);
```

```
break;
```

```
case 2:
```

```
first = delete_rear (first);
```

```
break;
```

```
case 3:
```

```
printf ("Enter item to be inserted at given position\n");
```

```
scanf ("%d", &item);
```

```
printf ("Enter position");
```

```
scanf ("%d", &pos);
```

```
first = insert_pos (item, pos, first);
```

```
break;
```

```
case 4: printf ("Enter the position\n");
```

```
scanf ("%d", &pos);
```

```
first = delete_pos (pos, first);
```

```
break;
```

```
case 5: display (first);
```

```
break;
```

```
default: exit (0);
```

```
break;
```

```
}
```

```
}
```

```
}
```

18M19A5090

RAHIL

```

1 #include<stdio.h>
2 struct node
3 {
4     int info;
5     struct node *link;
6 };
7 typedef struct node *NODE;
8 NODE getnode()
9 {
10     NODE x;
11     x=(NODE)malloc(sizeof(struct node));
12     if(x==NULL)
13     {
14         printf("Memory full\n");
15         exit(1);
16     }
17     return x;
18 }
19 void freenode(NODE x)
20 {
21     free(x);
22 }
23 NODE insert_front(NODE first,int item)
24 {
25     NODE temp;
26     temp=getnode();
27     temp->info=item;
28     temp->link=NULL;
29     if(first==NULL)
30     {
31         return temp;
32     }
33     temp->link=first;
34     first=temp;
35     return first;
36 }
37 NODE insert_rear(NODE first,int item)
38 {
39     NODE temp,cur;
40     temp=getnode();
41     temp->info=item;
42     temp->link=NULL;
43     if(first==NULL)
44     {
45         return temp;
46     }
47     cur=first;
48     while(cur->link!=NULL)
49     {
50         cur=cur->link;
51     }
52     cur->link=temp;
53     return first;
54 }
55 NODE delete_front(NODE first)
56 {
57     NODE temp;
58     if(first==NULL)
59     {
60         printf("List is Empty, Cannot Delete!\n");
61         return first;
62     }
63     temp=first;
64     temp->link=temp->link;
65     free(temp);
66     return temp;
67 }
68 NODE delete_rear(NODE first)
69 {
70     NODE cur,prev;
71     if(first==NULL)
72     {
73         printf("List is Empty, Cannot Delete!\n");
74         return first;
75     }
76     if(first->link==NULL)
77     {
78         printf("Item Deleted at Front-end = %d\n",first->info);
79         free(first);
80         return first;
81     }
82     prev=NULL;
83     cur=first;
84     while(cur->link!=NULL)
85     {
86         prev=cur;
87         cur=cur->link;
88     }
89     printf("Item Deleted at Rear-end = %d\n",cur->info);
90     free(cur);
91     prev->link=NULL;
92     return first;
93 }
94 NODE insert_pos(int item,int pos,NODE first)
95 {
96     NODE temp,cur,prev;
97     int count;
98     temp=getnode();
99     temp->info=item;
100     temp->link=NULL;
101     if(first==NULL && pos==1)
102     {
103         return temp;
104     }
105     if(first==NULL)
106     {
107         printf("Invalid Position!\n");
108         return first;
109     }
110     if(pos==1)
111     {
112         temp->link=first;
113         first=temp;
114         return temp;
115     }
116     count=1;
117     prev=NULL;
118     cur=first;
119     while((cur!=NULL && count!=pos))
120     {
121         prev=cur;
122         cur=cur->link;
123         count++;
124     }
125     if(count==pos)
126     {
127         prev->link=temp;
128         temp->link=cur;
129         return first;
130     }
131     printf("Invalid Position!\n");
132     return first;
133 }
134 NODE delete_pos(int pos,NODE first)
135 {
136     NODE cur;
137     NODE prev;
138     int count=1;
139     if(first==NULL || pos<1)
140     {
141         printf("Invalid Position!\n");
142         return NULL;
143     }
144     if(pos==1)
145     {
146         cur=first;
147         first=first->link;
148         free(cur);
149         return first;
150     }
151     prev=NULL;
152     cur=first;
153     while((cur!=NULL))
154     {
155         if(count==pos){flag=1;break;}
156         count++;
157         prev=cur;
158         cur=cur->link;
159     }
160     if(flag==1)
161     {
162         printf("Invalid Position!\n");
163         return first;
164     }
165     printf("Item Deleted at Given Position = %d\n",cur->info);
166     prev->link=cur->link;
167     free(cur);
168     return first;
169 }
170 void display(NODE first)
171 {
172     NODE temp;
173     if(first==NULL)
174     {
175         printf("Empty List, Cannot Display Items.\n");
176         return;
177     }
178     temp=first;
179     while(temp->link!=NULL)
180     {
181         printf("%d\n",temp->info);
182         temp=temp->link;
183     }
184 }
185 int main()
186 {
187     int item,choice,key,pos;
188     int count=0;
189     NODE first=NULL;
190     while(1)
191     {
192         printf("1.Insert_Front\n2.Insert_Rear\n3.Delete_Front\n4.Delete_Rear\n");
193         printf("5.Insert_Info_Position\n6.Delete_Info_Position\n7.Display_List\n8.Exit\n");
194         printf("Enter Choice : ");
195         scanf("%d",&choice);
196         switch(choice)
197         {
198             case 1:
199                 printf("Enter the item at Front-end : ");
200                 scanf("%d",&item);
201                 first=insert_front(first,item);
202                 break;
203             case 2:
204                 printf("Enter the item at Rear-end : ");
205                 scanf("%d",&item);
206                 first=insert_rear(first,item);
207                 break;
208             case 3:
209                 first=delete_front(first);
210                 break;
211             case 4:
212                 first=delete_rear(first);
213                 break;
214             case 5:
215                 printf("Enter Item to be Inserted at Given Position:");
216                 scanf("%d",&item);
217                 printf("Enter Position:");
218                 scanf("%d",&pos);
219                 first=insert_pos(item,pos,first);
220                 break;
221             case 6:
222                 printf("Enter Position:");
223                 scanf("%d",&pos);
224                 first=delete_pos(pos,first);
225                 break;
226             case 7:
227                 display(first);
228                 break;
229             default:
230                 exit(1);
231                 break;
232         }
233     }
234 }

```

```

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Insert_Info_Position
6.Delete_Info_Position
7.Display List
8.Exit
Enter Choice:1
Enter the Item at Front-end: 10

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Insert_Info_Position
6.Delete_Info_Position
7.Display List
8.Exit
Enter Choice:1
Enter the Item at Front-end: 20

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Insert_Info_Position
6.Delete_Info_Position
7.Display List
8.Exit
Enter Choice:1
Enter the Item at Front-end: 30

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Insert_Info_Position
6.Delete_Info_Position
7.Display List
8.Exit
Enter Choice:7
30
20
10

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Insert_Info_Position
6.Delete_Info_Position
7.Display List
8.Exit
Enter Choice:5
Enter Item to be Inserted at Given Position:25
Enter Position:2

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Insert_Info_Position
6.Delete_Info_Position
7.Display List
8.Exit
Enter Choice:6
Enter Position:4
Item Deleted at Given Position = 10

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Insert_Info_Position
6.Delete_Info_Position
7.Display List
8.Exit
Enter Choice:7
30
25
20

```