

7.1.1

```
C PriorityQ...
1  #include<stdio.h>
2  v struct node{
3      int data;
4      int priority;
5      struct node * next;
6  };
7  struct node * front = NULL;
8  v void enqueue(int data,int priority){
9      —> struct node * temp=(struct node*)malloc(sizeof(struct node));
10     —> temp->data=data;
11     —> temp->priority=priority;
12     —> temp->next=NULL;
13     v —> if(front==NULL || priority<front->priority){
14         —> —> temp->next=front;
15         —> —> front=temp;
16     —> }
17     v —> else{
18         —> —> struct node * cur=front;
19         v —> —> while(cur->next!=NULL&&cur->next->priority<=priority){
20             —> —> —> cur=cur->next;
21             —> —> }
22         —> —> temp->next=cur->next;
23         —> —> cur->next=temp;
24     —> }
25 }
26 v void deque(){
27     v —> if(front==NULL){
28         —> —> printf("Priority queue is underflow.\n");
29         —> —> return;
```

```

30     }
31     struct node * temp=front;
32     printf("Deleted value = %d\n",temp->data);
33     front=front->next;
34     free(temp);
35 }
36 void display(){
37     if(front==NULL){
38         printf("Priority queue is empty.\n");
39         return;
40     }
41     struct node * cur=front;
42     printf("Elements in the priority queue : ");
43     while(cur!=NULL){
44         printf("%d (%d) ",cur->data,cur->priority);
45         cur=cur->next;
46     }
47     printf("\n");
48 }
49 int size(){
50     int cnt=0;
51     struct node * cur=front;
52     while(cur!=NULL){
53         cnt++;
54         cur=cur->next;
55     }
56     return cnt;
57 }
58 int isEmpty(){
59     return front==NULL;

```

```

60     }
61     v int main(){
62         —>int choice,data,priority;
63     v —>while(1){
64         —>—>printf("1.Enqueue·2.Dequeue·3.Display·4.Is·Empty·5.Size·6.Exit\n");
65         —>—>printf("Enter·your·option·:·");
66         —>—>scanf("%d",&choice);
67     v —>—>switch(choice){
68         —>—>—>case·1:
69         —>—>—>—>printf("Enter·element·:·");
70         —>—>—>—>scanf("%d",&data);
71         —>—>—>—>printf("Enter·priority·:·");
72         —>—>—>—>scanf("%d",&priority);
73         —>—>—>—>enqueue(data,priority);
74         —>—>—>—>break;
75         —>—>—>case·2:
76         —>—>—>—>deque();
77         —>—>—>—>break;
78         —>—>—>case·3:
79         —>—>—>—>display();
80         —>—>—>—>break;
81         —>—>—>case·4:
82     v —>—>—>—>if(isEmpty()){
83         —>—>—>—>—>printf("Priority·queue·is·empty.\n");
84         —>—>—>—>}
85     v —>—>—>—>else{
86         —>—>—>—>—>printf("Priority·queue·is·not·empty.\n");
87         —>—>—>—>}
88         —>—>—>—>break;
89         —>—>—>case·5:
90         —>—>—>—>printf("Priority·queue·size·:·%d\n",size());
91         —>—>—>—>break;
92         —>—>—>case·6:
93         —>—>—>—>exit(0);
94         —>—>}
95     —>}
96     }

```

7.1.2

```

1  #include<stdio.h>
2  #include<stdlib.h>
3  struct node
4  v {
5      —>int data;
6      —>struct node *next;
7  };
8  struct node *root=NULL;
9  void ins_beg()
10 v {
11     —>struct node *temp,*p;
12     —>temp = (struct node*)malloc(sizeof(struct node));
13     —>printf("Enter an element : ");
14     —>scanf("%d",&temp->data);
15     —>temp->next = NULL;
16     —>if(root == NULL)
17 v —>{
18     —>—>root = temp;
19     —>}
20     —>else
21 v —>{
22     —>—>
23     —>—>temp->next = root;
24     —>—>root = temp;
25     —>}
26     }
27 void del()
28 v {
29     —>struct node *p;

```

```

30     —> p = root;
31     —> if(p == NULL)
32 v —> {
33     —> —> printf("Single Linked List is empty so deletion is not possible\n");
34     —> —>
35     —> }
36     —> else
37 v —> {
38     —> —>
39     —> —> root = root->next;
40     —> —> p->next = NULL;
41     —> —> printf("The deleted element from SLL : %d\n", p->data);
42     —> —> free(p);
43     —> }
44     }
45     void traverse()
46 v {
47     —>
48     —> struct node *p;
49     —> p = root;
50     —> if(p == NULL)
51 v —> {
52     —> —> printf("Single Linked List is empty\n");
53     —> }
54     —> else
55 v —> {
56     —> —> printf("The elements in SLL are : ");
57     —> —> while(p != NULL)
58 v —> —> {
59     —> —> —> printf("%d ", p->data);

```

```

60     p = p->next;
61 }
62 printf("NULL\n");
63
64
65 }
66 }
67 void reverse()
68 {
69     struct node *ptr1,*ptr2,*ptr3;
70     ptr1 = root;
71     ptr2 = NULL;
72     ptr3 = NULL;
73     while(ptr1!=NULL)
74     {
75         ptr3 = ptr1->next;
76         ptr1->next = ptr2;
77         ptr2 = ptr1;
78         ptr1 = ptr3;
79     }
80     root = ptr2;
81     printf("Single Linked List is reversed\n");
82 }
83 int main()
84 {
85     int op;
86     while(1)
87     {
88         printf("1.Insert-At-Begin-2.Delete-at-Begin-3.Reverse-the-list-4.Traverse-the-List-5.Exit\n");
89         printf("Enter your option : ");

```

```

90     —>scanf("%d",&op);
91     —>switch(op)
92 v —>{
93     —>—>case 1:
94     —>—>ins_beg();
95     —>—>break;
96     —>—>
97     —>—>case 2:
98     —>—>del();
99     —>—>break;
100    —>—>
101    —>—>case 3:
102    —>—>reverse();
103    —>—>break;
104    —>—>
105    —>—>case 4:
106    —>—>traverse();
107    —>—>break;
108    —>—>
109    —>—>case 5:
110    —>—>exit(0);
111    —>—>
112    —>}
113    —>—>
114    —>}
115    }

```

7.1.3

```

1  #include<stdio.h>
2  #include<stdlib.h>
3  struct node
4  v {
5      —>int data;
6      —>struct node *next;
7  };
8
9  typedef struct node *NODE;
10 NODE createNode()
11 v {
12     —>NODE temp;
13     —>temp = (NODE)malloc(sizeof(struct node));
14     —>temp->next = NULL;
15     —>return temp;
16 }
17 NODE insertAtBegin(NODE head,int x)
18 v {
19     —>NODE temp;
20     —>temp = createNode();
21     —>temp->data = x;
22     —>temp->next = head;
23     —>head = temp;
24     —>return head;
25 }
26 int searchPosOfEle(NODE head,int key)
27 v {
28     —>NODE currentNode = head;
29     —>int pos = 0;

```



```

30     —> if(currentNode == NULL)
31     v —> {
32         —> return pos;
33     —> }
34     —> while(currentNode != NULL && currentNode->data != key)
35     v —> {
36         —> if(currentNode->next == NULL)
37     v —> {
38         —> return 0;
39     —> }
40     —> pos++;
41     —> currentNode = currentNode->next;
42     —> }
43     —> return (pos+1);
44     }
45     void traverseList(NODE head)
46     v {
47         —> NODE temp = head;
48         —> while(temp != NULL)
49     v —> {
50     —> printf("%d --> ", temp->data);
51     —> temp = temp->next;
52     —> }
53     —> printf("NULL\n");
54     }

```

7.1.4

```

1  #include<stdio.h>
2  #include<stdlib.h>
3
4  v struct node {
5      —>int data;
6      —>struct node * next;
7  };
8
9  typedef struct node * NODE;
10
11 v NODE createAndAddNodes(NODE first) {
12     —>NODE temp, q;
13     —>int x;
14     —>printf("Enter element : ");
15     —>scanf("%d", &x);
16 v —>while (x != -1) {
17     —>—>temp = (NODE) malloc(sizeof(struct node));
18     —>—>temp->data = x;
19     —>—>temp->next = NULL;
20 v —>—>if (first == NULL) {
21     —>—>—>first = temp;
22 v —>—>} else {
23     —>—>—>q->next = temp;
24     —>—>}
25     —>—>q = temp;
26     —>—>printf("Enter element : ");
27     —>—>scanf("%d", &x);
28     —>}
29     —>return first;

```

```

30     }
31     v NODE.merge(NODE.t1, NODE.t2) {
32         —>
33         —> if (t1 == NULL) return t2;
34         —> if (t2 == NULL) return t1;
35     v —> if (t1->data < t2->data) {
36         —> —> t1->next = merge(t1->next, t2);
37         —> —> return t1;
38     v —> } else {
39         —> —> t2->next = merge(t1, t2->next);
40         —> —> return t2;
41     —> }
42     }
43
44     v NODE.sort(NODE.first) {
45         —> NODE.t1, t2;
46         —> int x;
47         —>
48     v —> for (t1 = first; t1->next != NULL; t1 = t1->next) {
49     v —> —> for (t2 = t1->next; t2 != NULL; t2 = t2->next) {
50     v —> —> —> if (t1->data > t2->data) {
51         —> —> —> —>
52         —> —> —> —> x = t1->data;
53         —> —> —> —> t1->data = t2->data;
54         —> —> —> —> t2->data = x;
55         —> —> —> —>
56         —> —> —> }
57     —> —> }
58     —> }

```

```

59     —> return first;
60     }
61     v void print(NODE node) {
62     v —> while (node != NULL) {
63         —> —> printf("%d ", node->data);
64         —> —> node = node->next;
65     —> }
66     }
67
68

```

7.2.1

```

1  #include<stdio.h>
2  int main()
3  v {
4      —>int n,arr[100];
5      —>printf("Enter the size of the array: ");
6      —>scanf("%d",&n);
7      —>heap_sort(arr,n);
8  }
9
10
11
12
13 v void heap_sort(int arr[], int n){
14     —>
15     int i,j;
16     —>printf("Enter elements to sort: ");
17     —>for(i=0;i<n;i++)
18 v —>{
19     —>—>scanf("%d",&arr[i]);
20     —>}
21     —>printf("The initial array is: ");
22     —>for(i=0;i<n;i++)
23 v —>{
24     —>—>printf("%d ",arr[i]);
25     —>}
26     —>printf("\n");
27     —>for(i=0;i<n;i++)
28 v —>{
29     —>—>for(j=i+1;j<n;j++)

```

```

30  v  —>{
31      —>{if(arr[i]<arr[j])
32  v  —>{
33      —>{int temp = arr[i];
34      —>{arr[i] = arr[j];
35      —>{arr[j] = temp;
36      —>}
37      —>}
38      —>}
39      —>
40      —>printf("The sorted array is: ");
41      —>for(i=0;i<n;i++)
42  v  —>{
43      —>{printf("%d.",arr[i]);
44      —>}
45      —>printf("\n");
46      —>
47      }

```

7.2.2

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  v typedef struct node {
5      —>int data;
6      —>struct node* next;
7  } Node;
8
9  v Node* createNode(int data) {
10     —>Node* newNode = (Node*)malloc(sizeof(Node));
11     v —>if (!newNode) {
12         —>—>printf("Memory error\n");
13         —>—>return NULL;
14     —>}
15     —>newNode->data = data;
16     —>newNode->next = NULL;
17     —>return newNode;
18 }
19 v Node* insertNode(Node* head, int data) {
20     —>Node* newNode = createNode(data);
21     —>if (!newNode) return head;
22     v —>if (head == NULL) {
23
24         —>—>head = newNode;
25     v —>} else {
26         —>—>Node* current = head;
27     v —>—>while (current->next != NULL) {
28         —>—>—>current = current->next;
29     —>—>}

```

```

30     —> —>current->next := newNode;
31     —>}
32     —>return head;
33 }
34 v void findNthFromEnd(Node* head, int n) {
35     —>Node* main_ptr := head;
36     —>Node* ref_ptr := head;
37     —>int count := 0;
38 v —>if (head != NULL) {
39 v —> —>while (count < n) {
40 v —> —> —>if (ref_ptr == NULL) {
41     —> —> —>printf("-1\n");
42     —> —> —>return;
43     —> —> —>}
44     —> —> —>ref_ptr := ref_ptr->next;
45     —> —> —>count++;
46     —> —>}
47 v —> —>if (ref_ptr == NULL) {
48     —> —> —>printf("-1\n");
49 v —> —>} else {
50 v —> —> —>while (ref_ptr != NULL) {
51     —> —> —> —>main_ptr := main_ptr->next;
52     —> —> —> —>ref_ptr := ref_ptr->next;
53     —> —> —>}
54     —> —> —> —>printf("%d\n", main_ptr->data);
55     —> —>}
56     —>}
57 }
58 v int main() {
59     —>Node* head := NULL;

```

```

60     —>int n, data, N;
61     —>printf("Enter the number of nodes in the linked list: ");
62     —>scanf("%d", &n);
63     —>printf("Enter the values for nodes: ");
64 v —>for (int i := 0; i < n; i++) {
65     —> —>scanf("%d", &data);
66     —> —>head := insertNode(head, data);
67     —>}
68     —>printf("Enter the value of N: ");
69     —>scanf("%d", &N);
70     —>findNthFromEnd(head, N);
71
72
73 }

```