

线性逻辑结构

相关概念

线性结构特点

线性表基本操作

线性表的存储结构

顺序存储结构

静态分配的顺序表存储

动态分配的顺序表存储

线性表的链式存储结构

单链表

不带头结点的单链表

带头结点的单链表

静态链表

双向循环链表（带头结点）

## 线性逻辑结构

线性表： $n$  ( $n > 0$ ) 个数据元素的有限序列

## 相关概念

直接前驱

直接后继

长度

## 线性结构特点

在数据元素的非空有限集中：

1. 存在唯一——一个被称为“第一个”的数据元素；
2. 存在唯一——一个被称为“最后一个”的数据元素；
3. 除第一个元素外，每个数据元素均有唯一——一个直接前驱；
4. 除最后一个元素外，每个数据元素均有唯一——一个直接后继；

## 线性表基本操作

增：

- 初始化结构
- 插入元素

删：

- 回收结构
- 清空线性表
- 删除结点

查：

- 线性表是否为空
- 线性表的长度
- 求第*i*个元素
- 求函数值*x*在线性表中的位置
- 求函数值*x*在线性表中的直接前驱（直接后继）
- 遍历线性表

改：

- 在查的基础上进行更改

## 线性表的存储结构

### 顺序存储结构

顺序存储：将线性表中的元素相继存放在一个连续的存储空间中

### 静态分配的顺序表存储

```
1.  #define ListSize 100
2.  typedef int ListData;
3.
4.  //线性表的结构定义，数组从0开始
5.  typedef struct{
6.      ListData data[ListSize];
7.      int length;
8.  } SeqList;
9.
10. //初始化线性表结构
11. void initList(SeqList &l){
12.     l.length = 0;
13. }
14.
15. //销毁线性结构
16. void destroyList(SeqList &l){
17.     l.length = 0;
18. }
19.
20. //清空线性结构中的元素
21. void clearList(SeqList &l){
22.     l.length = 0;
23. }
24.
25. // 线性表是否为空
26. int listEmpty(SeqList l){
27.     return (!l.length);
28. }
29.
30. // 求线性表的长度
31. int listLength(SeqList l){
32.     return l.length;
33. }
34.
35. // 获得线性表中第i个元素的值
36. ListData getElem(SeqList l, int i){
37.     if(i>=1&&i<=l.length){
38.         return l.data[i-1];
39.     }
40.     else{
41.         return NULL;
42.     }
43. }
44.
45. // 定位结点在线性表中的位置序号
46. // 如果结点存在线性表中，返回结点位置，否则返回-1
47. int locateElem(SeqList l, ListData data){
48.     int count = 1;
49.     while((count<=l.length)&&(l.data[count-1]!=data)){
50.         count++;
51.     }
52.     if(count<=l.length){
53.         return count;
```

```

54.     }
55.     else{
56.         return -1;
57.     }
58. }
59.
60. // 查看结点是否存在线性表中，若存在，则返回1，构造返回0
61. int isIn(SeqList l, ListData data){
62.     int count = 1;
63.     while((count<=l.length)&&(l.data[count-1]!=data)){
64.         count++;
65.     }
66.     if(count<=l.length){
67.         return 1;
68.     }
69.     else{
70.         return 0;
71.     }
72. }
73.
74. // 定位元素在线性表中的直接后继
75. // 若结点的直接后继存在，则返回其直接后继的位置，否则返回-1
76. int nextElem(SeqList l, ListData data){
77.     int count = 1;
78.     while((count<=l.length-1)&&(l.data[count-1]!=data)){
79.         count++;
80.     }
81.     if(count<=l.length-1){
82.         return (count+1);
83.     }
84.     else{
85.         return -1;
86.     }
87. }
88.
89. // 定位元素在线性表中的直接前驱
90. // 若结点的直接前驱 存在，则返回其直接前驱额位置，否则返回-1
91. int priorElem(SeqList l, ListData data){
92.     int count= 2;
93.     while((count<=l.length)&&(l.data[count-1]!=data)){
94.         count++;
95.     }
96.     if(count<=l.length){
97.         return (count-1);
98.     }
99.     else{
100.         return -1;
101.     }
102. }
103.
104. // 将结点插入到线性表中的第i个位置（在原线性表的第i个位置之前插入）
105. // 插入成功则返回1，不成功返回-1
106. int listInsert(SeqList &l, int i, ListData data){
107.     if((i<1)||i>(l.length+1)|| (l.length==ListSize)){

```

```

108.         return -1;
109.     }
110.     else{
111.         for(int j=l.length;j>=i;j--){
112.             l.data[j] = l.data[j-1];
113.         }
114.         l.data[i-1] = data;
115.         l.length++;
116.         return 1;
117.     }
118. }
119.
120. // 删除线性表中的第i个元素
121. // 删除 成功返回1, 否则返回-1
122. int listDelete(SeqList &l, int i){
123.     if((i<1)|| (i>l.length)){
124.         return -1;
125.     }
126.     else{
127.         for(int j=i+1;j<=l.length;j++){
128.             l.data[j-2] = l.data[j-1];
129.         }
130.         l.length--;
131.         return 1;
132.     }
133. }

```

## 动态分配的顺序表存储

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.
4.  #define LIST_INIT_SIZE 10
5.  #define LISTINCREMENT 5
6.
7.  typedef int ListData;
8.
9.  //定义动态存储结构
10. typedef struct{
11.     ListData *data;
12.     int length;
13.     int listSize;
14. }SqList;
15.
16. //初始化动态存储结构
17. int initList(SqList &L){
18.     L.data = (ListData*)malloc(LIST_INIT_SIZE*sizeof(ListData));
19.     if(!L.data){
20.         exit(0);
21.     }
22.     L.length = 0;
23.     L.listSize = LIST_INIT_SIZE;
24.     return 1;
25. }
26.
27. //销毁动态存储结构
28. int destroyList(SqList &L){
29.     free(L.data);
30.     L.length = 0;
31.     L.listSize = 0;
32.     return 1;
33. }
34.
35. //清空动态存储结构
36. int clearList(SqList &L){
37.     L.length = 0;
38.     return 1;
39. }
40.
41. //是否为空
42. int listEmpty(SqList L){
43.     return (L.length == 0);
44. }
45.
46. //返回长度
47. int listLength(SqList L){
48.     return L.length;
49. }
50.
51. //得到第i个元素
52. ListData getElem(SqList L, int i){
53.     if(i<1||i>L.length){
```

```

54.         return -1;
55.     }
56.     return *(L.data+i-1);
57. }
58.
59. //获得元素的位置
60. int locateElem(SqList L, ListData data){
61.     int count = 1;
62.     while((count<=L.length)&&(*(L.data+count-1)!=data)){
63.         count++;
64.     }
65.     if(count==L.length+1){
66.         return -1;
67.     }
68.     return count;
69. }
70.
71. //判断是否在内
72. int isIn(SqList L, ListData data){
73.     int count = 1;
74.     while((count<L.length+1)&&(*(L.data+count-1)!=data)){
75.         count++;
76.     }
77.     if(count==L.length+1){
78.         return -1;
79.     }
80.     return 1;
81. }
82.
83. //求直接后继
84. int nextElem(SqList L, ListData data){
85.     int count = 1;
86.     while((count<L.length)&&(*(L.data+count-2)!=data)){
87.         count++;
88.     }
89.     if(count==L.length){
90.         return -1;
91.     }
92.     else{
93.         return count;
94.     }
95. }
96.
97. //求直接前驱
98. int priorElem(SqList L, ListData data){
99.     int count = 1;
100.    while((count < L.length-1)&&(*(L.data+count)!=data)){
101.        count++;
102.    }
103.    if(count==L.length-1){
104.        return -1;
105.    }
106.    else{
107.        return count;

```

```

108.     }
109. }
110.
111. //插入元素
112. int listInsert(SqlList &L, int i, ListData data){
113.     if(i<1||i>L.length+1){
114.         return -1;
115.     }
116.     if(L.length == L.listSize){
117.         int *newList = (ListData*)realloc(L.data,(L.listSize+LISTINCREMENT)*s
118.         if(!newList){
119.             exit(0);
120.         }
121.         L.data = newList;
122.         L.listSize = L.listSize + LISTINCREMENT;
123.     }
124.     for(int j = L.length;j>i-1;j--){
125.         *(L.data+j) = *(L.data+j-1);
126.     }
127.     *(L.data+i-1) = data;
128.     L.length++;
129.     return 1;
130. }
131.
132. //删除元素
133. int deleteElem(SqlList &L, int i){
134.     if(i<1||i>L.length){
135.         return -1;
136.     }
137.     for(i;i<L.length;i++){
138.         *(L.data+i-1) = *(L.data+i);
139.     }
140.     L.length--;
141.     return 1;
142. }

```

## 线性表的链式存储结构

- 单链表
- 静态链表
- 循环链表
- 双向链表

### 单链表

单链表：用一组地址任意的存储单元存放线性表中的数据元素

单链表结构：每个元素由结点构成，包括两个域：数据域Data和指针域next



## 不带头结点的单链表

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.
4.  typedef int ListData;
5.
6.  typedef struct node{
7.      ListData data;
8.      node* next;
9.  }ListNode;
10.
11. typedef ListNode* LinkList;
12.
13. //初始化单链表
14. void initList(LinkList &first){
15.     first = NULL;
16. }
17.
18. //销毁单链表
19. void destroyList(LinkList &first){
20.     ListNode* tem;
21.     while(first){
22.         tem = first;
23.         first = first->next;
24.         free(tem);
25.     }
26. }
27.
28. //清空单链表
29. void clearList(LinkList &first){
30.     ListNode* tem;
31.     while(first){
32.         tem = first;
33.         first = first->next;
34.         free(tem);
35.     }
36. }
37.
38. //链表是否为空
39. int listEmpty(LinkList &first){
40.     return (first==NULL);
41. }
42.
43. //求链表的长度
44. int listLength(LinkList first){
45.     int count = 1;
46.     ListNode* tem = first;
47.     while(tem){
48.         tem = tem->next;
49.         count++;
50.     }
51.     return count-1;
52. }
53.
```

```

54. //获得链表的第i个元素
55. ListData getElem(LinkList &first, int i){
56.     ListNode *tem = first;
57.     int count = 1;
58.     while(tem&&count<i){
59.         tem=tem->next;
60.         count++;
61.     }
62.     if(!tem||count > i){
63.         return NULL;
64.     }
65.     return tem->data;
66.
67. }
68.
69. //定位结点
70. int locateElem(LinkList first, ListData data){
71.     int count = 1;
72.     ListNode* tem = first;
73.     while(tem&&tem->data!=data){
74.         tem = tem->next;
75.         count++;
76.     }
77.     if(!tem){
78.         return -1;
79.     }
80.     return count;
81. }
82.
83. //是否在内
84. int isIn(LinkList first, ListData data){
85.     ListNode* tem = first;
86.     while(tem&&tem->data!=data){
87.         tem=tem->next;
88.     }
89.     if(!tem){
90.         return -1;
91.     }
92.     return 1;
93. }
94.
95. //求直接后继
96. ListData nextElem(LinkList first, ListData data){
97.     ListNode* tem = first;
98.     while(tem&&tem->next&&tem->data!=data){
99.         tem = tem->next;
100.    }
101.    if(!tem||!tem->next){
102.        return NULL;
103.    }
104.    return tem->next->data;
105. }
106.
107. //求直接前驱

```

```

108. ListData priorElem(LinkList first, ListData data){
109.     ListNode* tem = first;
110.     while(tem&&tem->next&&tem->next->data!=data){
111.         tem = tem->next;
112.     }
113.     if(!tem||!tem->next){
114.         return NULL;
115.     }
116.     return tem->data;
117. }
118.
119. //插入结点
120. int listInsert(LinkList &first, int i, ListData data){
121.     int count = 1;
122.     ListNode* tem = first;
123.     if(i == 1){
124.         ListNode* newNode = (ListNode*)malloc(sizeof(ListNode));
125.         newNode->data = data;
126.         newNode->next = first;
127.         first = newNode;
128.         return 1;
129.     }
130.     while(tem&&count<i-1){
131.         tem = tem->next;
132.         count++;
133.     }
134.     if(!tem||count>i){
135.         return -1;
136.     }
137.     ListNode* newNode = (ListNode*)malloc(sizeof(ListNode));
138.     newNode->data = data;
139.     newNode->next = tem->next;
140.     tem->next = newNode;
141.     return 1;
142. }
143.
144. //删除结点
145. int listDelete(LinkList &first, int i){
146.     if(first&&i == 1){
147.         ListNode* freeNode = first;
148.         first = first->next;
149.         free(freeNode);
150.         return 1;
151.     }
152.     int count = 1;
153.     ListNode* tem = first;
154.     while(tem&&count<i-1){
155.         tem = tem->next;
156.         count++;
157.     }
158.     if(!tem||!(tem->next)||i<1){
159.         return -1;
160.     }
161.     ListNode* freeNode = tem->next;

```

```
162.     tem->next = freeNode->next;
163.     free(freeNode);
164.     return 1;
165. }
```

## 带头结点的单链表

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.
4.  typedef int ListData;
5.
6.  typedef struct node{
7.      ListData data;
8.      node* next;
9.  }ListNode;
10.
11. typedef ListNode* LinkList;
12.
13. void initList(LinkList &first){
14.     first = (LinkList)malloc(sizeof(ListNode));
15.     if(!first){
16.         exit(0);
17.     }
18.     first->next = NULL;
19. }
20.
21. void destroyList(LinkList &first){
22.     ListNode* tem = first;
23.     while(first){
24.         tem = first;
25.         first = first->next;
26.         free(tem);
27.     }
28. }
29.
30. void clearList(LinkList &first){
31.     ListNode* tem = first->next;
32.     ListNode* freeNode = first->next;
33.     while(tem){
34.         freeNode = tem;
35.         tem = freeNode->next;
36.         free(freeNode);
37.     }
38.     first->next = NULL;
39. }
40.
41. int listEmpty(LinkList first){
42.     return (first->next==NULL);
43. }
44.
45. int listLength(LinkList first){
46.     int count = 1;
47.     ListNode* node = first->next;
48.     while(node){
49.         node=node->next;
50.         count++;
51.     }
52.     return (count-1);
53. }
```

```

54.
55. ListData getElem(LinkList first, int i){
56.     int count = 1;
57.     ListNode* node = first->next;
58.     while(node&&count<i){
59.         node=node->next;
60.         count++;
61.     }
62.     if(!node||count>i){
63.         return NULL;
64.     }
65.     return node->data;
66. }
67.
68. int locateElem(LinkList first, ListData data){
69.     int count = 1;
70.     ListNode* node = first->next;
71.     while(node&&node->data!=data){
72.         node = node->next;
73.         count++;
74.     }
75.     if(!node){
76.         return NULL;
77.     }
78.     return count;
79. }
80.
81. int isIn(LinkList first, ListData data){
82.     ListNode *node = first->next;
83.     while(node&&node->data!=data){
84.         node = node->next;
85.     }
86.     if(!node){
87.         return -1;
88.     }
89.     return 1;
90. }
91.
92. ListData nextElem(LinkList first, ListData data){
93.     ListNode* node = first->next;
94.     while(node&&node->next&&node->data!=data){
95.         node = node->next;
96.     }
97.     if(!node||!node->next){
98.         return NULL;
99.     }
100.    return node->next->data;
101. }
102.
103. ListData priorElem(LinkList first, ListData data){
104.     ListNode* node = first->next;
105.     while(node&&node->next&&node->next->data!=data){
106.         node = node->next;
107.     }

```

```

108.     if(!node||!node->next){
109.         return NULL;
110.     }
111.     return node->data;
112. }
113.
114. int listInsert(LinkList first, int i, ListData data){
115.     ListNode* node = first;
116.     int count = 0;
117.     while(node&&count < i-1){
118.         node = node->next;
119.         count++;
120.     }
121.     if(!node||count>i-1){
122.         return -1;
123.     }
124.     ListNode* newNode = (ListNode*)malloc(sizeof(ListNode));
125.     newNode->data = data;
126.     newNode->next = node->next;
127.     node->next = newNode;
128.     return 1;
129. }
130.
131.
132. int listDelete(LinkList first, int i){
133.     int count = 0;
134.     ListNode* node = first;
135.     while(node&&node->next&&count<i-1){
136.         node = node->next;
137.         count++;
138.     }
139.     if(!node||!node->next||i<1){
140.         return -1;
141.     }
142.
143.     ListNode* freeNode = node->next;
144.     node->next = freeNode->next;
145.     free(freeNode);
146.     return 1;
147. }

```

## 静态链表



```

1.  const int MaxSize = 100;
2.  typedef int ListData;
3.  typedef struct node{
4.      ListData data;
5.      int link;
6.  }SNode;
7.
8.  typedef struct{
9.      SNode Nodes[MaxSize];
10.     in newptr;
11. }SLinkList;
12.
13. void initSpace(SLinkList &space){
14.     space.newptr = 0;
15.     for(int i=0;i<MaxSize-1;i++){
16.         space.Nodes[i].link = i+1;
17.     }
18.     space.Nodes[MaxSize-1].link = -1;
19. }
20.
21. void initList(SLinkList &space, int SLink){
22.     if(space.newptr!=-1){
23.         SLink = space.newptr;
24.         space.newptr = space.Nodes[space.newptr].link;
25.         space.Nodes[SLink].link = -1;
26.     }
27. }
28.
29. void destroyList(SLinkList &space, int SLink){
30.     int tem = SLink;
31.     while(space.Nodes[tem].link!=-1){
32.         tem = space.Nodes[tem].link;
33.     }
34.     space.Nodes[tem].link = space.newptr;
35.     space.newptr = SLink;
36.     SLink = -1;
37. }
38.
39. void clearList(SLinkList &space, int SLink){
40.     int tem = space.Nodes[SLink].link;
41.     while(space.Nodes[tem].link!=-1){
42.         tem = space.Nodes[tem].link;
43.     }
44.     space.Nodes[tem].link = space.newptr;
45.     space.newptr = space.Nodes[SLink].link;
46.
47.     space.newptr[SLink].link = -1;
48. }
49.
50. int listEmpty(SLinkList &space, int SLink){
51.     return (space.Nodes[SLink].link==-1);
52. }
53.

```

```

54.
55. int listLength(SLinkList space, int SLink){
56.     int count = 0;
57.     int tem = SLink;
58.     while(space.Nodes[tem].link!=-1){
59.         tem = space.Nodes[tem].link;
60.         count++;
61.     }
62.     return count;
63. }
64.
65. ListData getElem(SLinkList space, int SLink, int i){
66.     int count = 0;
67.     int tem = SLink;
68.     if(SLink<0||SLink>MaxSize-1){
69.         return NULL;
70.     }
71.     while(space.Nodes[tem].link!=-1&&count<i){
72.         tem = space.Nodes[tem].link;
73.         count++;
74.     }
75.     if(space.Nodes[tem].link==--1||count>i){
76.         return NULL;
77.     }
78.     return space.Nodes[count].data;
79. }
80.
81.
82. //特殊，返回的不是元素在链表中的位置，而是元素在静态数组中的位置
83. int locateElem(SLinkList space, int SLink, ListData data){
84.     int count = 0;
85.     int tem = SLink;
86.     while(space.Nodes[tem].link!=-1&&space.Nodes[tem].data!=data){
87.         tem = space.Nodes[tem].link;
88.         count++;
89.     }
90.     if(space.Nodes[tem].link==--1){
91.         return NULL;
92.     }
93.     return tem;
94. }
95.
96. int isIn(SLinkList space, int SLink, ListData data){
97.     int tem = SLink;
98.     while(tem!=-1&&space.Nodes[tem].data!=data){
99.         tem = space.Nodes[tem].link ;
100.    }
101.    if(tem==--1){
102.        return -1;
103.    }
104.    return tem;
105. }
106.
107. int nextElem(SLinkList space, int SLink, ListData data){

```

```

108.     int tem = SLink;
109.     while(tem!=-1&&space.Nodes[tem].link!=-1&&space.Nodes[tem].data!=data){
110.         tem = space.Nodes[tem].link;
111.     }
112.     if(tem==-1 || space.Nodes[tem].link==-1){
113.         return -1;
114.     }
115.     return space.Nodes[tem].link
116. }
117.
118. int priorElem(SLinkList space, int SLink, ListData data){
119.     int tem = SLink;
120.     while(tem!=-1&&space.Nodes[tem].link!=-1&&space.Nodes[space.Nodes[tem].li
121.         tem = space.Nodes[tem].link;
122.     }
123.     if(tem==-1 || space.Nodes[tem].link==-1){
124.         return -1;
125.     }
126.     return tem;
127. }
128.
129. int listInsert(SLinkList &space, int SLink, int i, ListData data){
130.     int count = 0;
131.     int tem = SLink;
132.     while(tem!=-1&&count<i-1){
133.         tem = space.Nodes[tem].link;
134.     }
135.     if(tem==-1 || count>i-1){
136.         return -1;
137.     }
138.     if(space.newptr!=-1){
139.         int newNode = space.newptr;
140.         space.Nodes[newNode].data = data;
141.         space.Nodes[newNode].link = space.Nodes[tem].link;
142.         space.Nodes[tem].link = newNode;
143.         return 1
144.     }
145.     else{
146.         return -1;
147.     }
148. }
149.
150. int listDelete(SLinkList &space, int SLink, ListData data){
151.     int count = 0;
152.     int tem = SLink;
153.     while(tem!=-1&&space.Nodes[tem].link!=-1&&space.Nodes[space.Nodes[tem].li
154.         tem = space.Nodes[tem].link;
155.     }
156.     if(tem==-1 || space.Nodes[tem].link==-1){
157.         return -1;
158.     }
159.     int freeNode = space.Nodes[tem].link;
160.     space.Nodes[tem].link = space.Nodes[freeNode].link;
161.     space.Nodes[freeNode].link = space.newptr;

```

```
162.     space.newptr = freeNode;  
163.     return 1;  
164. }
```

## 双向循环链表（带头结点）

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.
4.  typedef int ListData;
5.  typedef struct DNode{
6.      ListData data;
7.      struct DNode *prior, *next;
8.  } DNode;
9.
10. typedef DNode* DList;
11.
12. void initList(DList &l){
13.     l = (DNode*)malloc(sizeof(DNode));
14.     if(!l){
15.         printf("存储分配出错! \n");
16.         exit(1);
17.     }
18.     l->prior = l;
19.     l->next = l;
20. }
21.
22. void clearList(DList &l){
23.     DNode *node = l->next;
24.     while(node!=l){
25.         DNode *tem = node;
26.         node = node->next;
27.         free(tem);
28.     }
29. }
30.
31. void destroyList(DList &l){
32.     DNode *node = l->next;
33.     while(node!=l){
34.         DNode *tem = node;
35.         node = node->next;
36.         free(tem);
37.     }
38.     free(l);
39. }
40.
41. int listEmpty(DList l){
42.     return (l->next==l);
43. }
44.
45. void listLength(DList l){
46.     DNode *node = l->next;
47.     int count = 0;
48.     while(node!=l){
49.         node = node->next;
50.         count++;
51.     }
52. }
53.
```

```

54. ListData getElem(DList l, int i){
55.     DNode *node = l->next;
56.     int count = 1;
57.     while(node!=l&&count<=i){
58.         node = node->next;
59.         count++;
60.     }
61.     if(count>i||node==l){
62.         return NULL;
63.     }
64.     else{
65.         return node->data;
66.     }
67. }

68.
69. int locate(DList l, ListData data){
70.     DNode *node = l->next;
71.     int count = 0;
72.     while(node!=l&&node->data!=data){
73.         count++;
74.         node = node->next;
75.     }
76.     if(node==l){
77.         return -1;
78.     }
79.     else{
80.         return count;
81.     }
82. }

83.
84. int isIn(DList l, ListData data){
85.     DNode *node = l->next;
86.     while(node!=l&&node->data!=data){
87.         node = node->next;
88.     }
89.     if(node==l){
90.         return -1;
91.     }
92.     else{
93.         return 1;
94.     }
95. }

96.
97. DNode* nextElem(DList l, ListData data){
98.     DNode *node = l->next;
99.     while(node!=l&&node->data!=data){
100.         node = node->next;
101.     }
102.     if(node==l||node->next==l){
103.         return NULL;
104.     }
105.     return node->next;
106. }
107.

```

```

108. DNode* priorElem(DList l, ListData data){
109.     DNode *node = l->next;
110.     while(node!=l&&node->next->data!=data){
111.         node = node->next;
112.     }
113.     if(node==l || node->next==l){
114.         return NULL;
115.     }
116.     return node;
117. }
118.
119. int insert(DList l, ListData data, int i){
120.     int count = 1;
121.     DNode *node = l->next;
122.     while(node!=l&&count<i){
123.         count++;
124.         node = node->next;
125.     }
126.     if(node==l || count>=i){
127.         return 0;
128.     }
129.     else{
130.         DNode *newNode = (DNode*)malloc(sizeof(DNode));
131.         newNode->data = data;
132.         newNode->next = node->next;
133.         node->next->prior = newNode;
134.         newNode->prior = node;
135.         node->next = newNode;
136.         return 1;
137.     }
138. }
139.
140. int delete(DList l, int i){
141.     int count=1;
142.     DNode *node = l->next;
143.     while(node!=l&&count<=i){
144.         count++;
145.         node = node->next;
146.     }
147.     if(node == l || count>i){
148.         return 0;
149.     }
150.     node->prior->next = node->next;
151.     node->next->prior = node->prior;
152.     free(node);
153. }

```