

| |
|---------------|
| 1.双向链表 |
| 1.1双向链表的特点 |
| 1.2双向链表的结构 |
| 1.3双向链表的常见操作 |
| 1.3.1双向链表的创建 |
| 1.3.2双向链表头插 |
| 1.3.3双向链表尾插 |
| 1.3.4双向链表位置插 |
| 1.3.5双向链表的遍历 |
| 1.3.6双向链表判空 |
| 1.3.7双向链表的头删 |
| 1.3.8双向链表的尾删 |
| 1.3.9双向链表的位置删 |
| 1.3.10双向链表的查询 |
| 1.3.11双向链表的更新 |
| 1.3.12双向链表的逆序 |
| 1.4整体代码 |
| Dplist.h |
| Dplist.c |
| main.c |

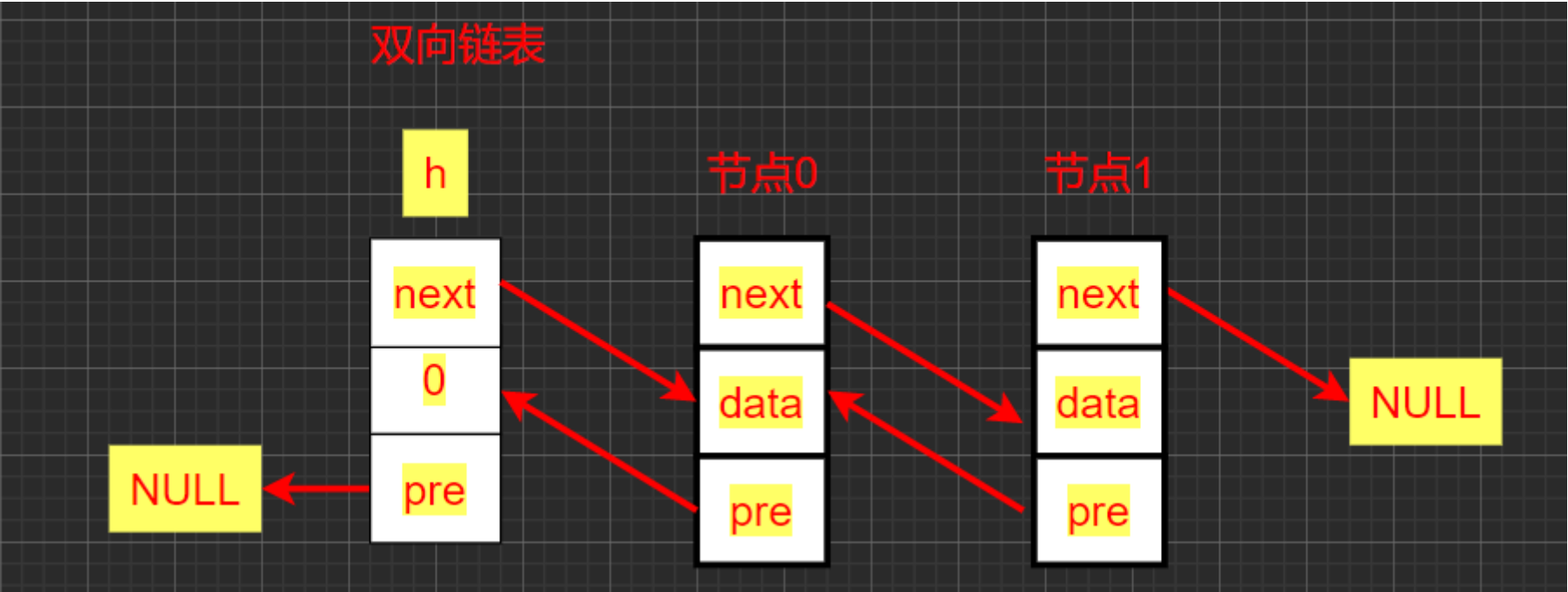
1.双向链表

1.1双向链表的特点

双向链表：对于单链表来说如果站在其中一个节点找前面的节点是无法实现的，对于单向循环链表可以实现，但是时间复杂度高。为了解决这一问题引入了双向链表，对应双向链表来说，每个节点有一个数据域和两个指针（pre,next）域，pre向前指的指针，next向后指的指针。

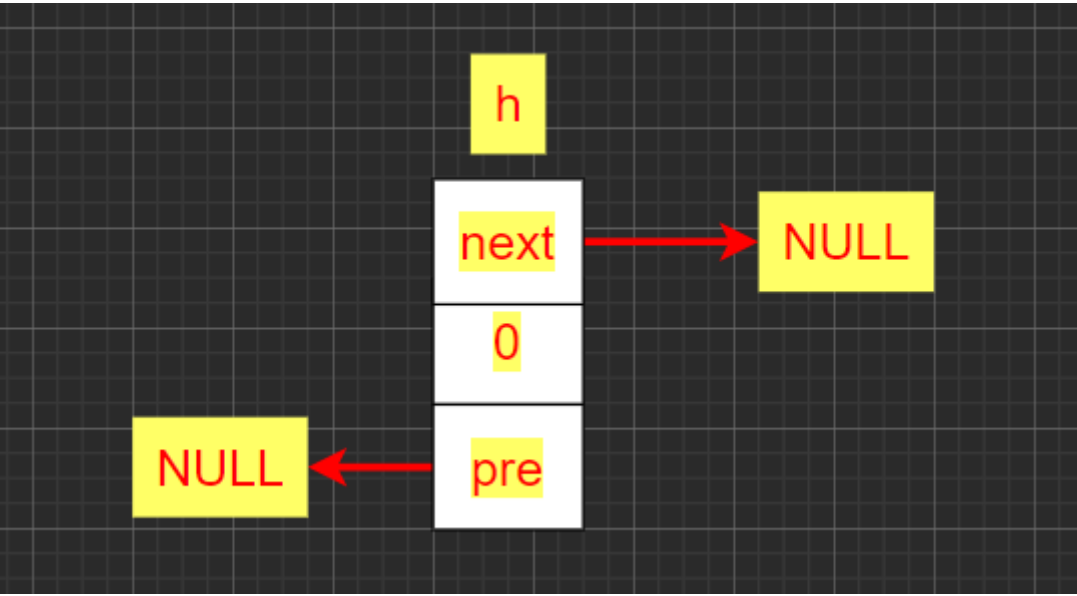
1.2双向链表的结构

```
1 #define datatype int
2 typedef struct node{
3     datatype data;
4     struct node *pre,*next;
5 }Dplist_t;
```



1.3双向链表的常见操作

1.3.1双向链表的创建

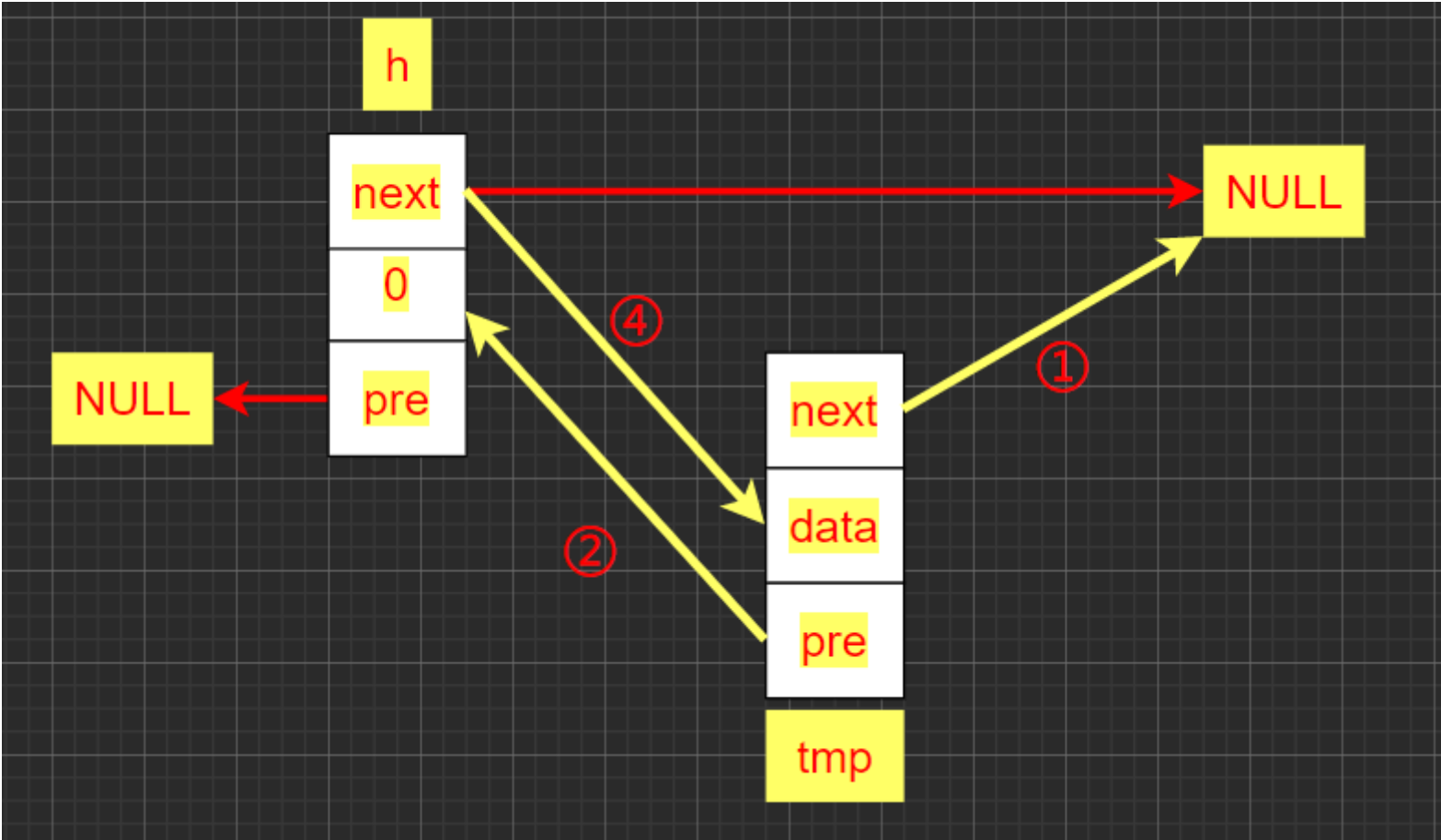


```
1 DPlist_t* DPListCreate(void)
2 {
3     DPlist_t* h;
4     h = (DPlist_t*)malloc(sizeof(*h));
5     if (h == NULL) {
6         printf("%s malloc memory error\n", __func__);
7         return NULL;
8     }
9     h->data = (datatype)0;
10    h->next = NULL;
11    h->pre = NULL;
12
13    return h;
14 }
```

1.3.2双向链表头插

情况1：（双链表是空链表）

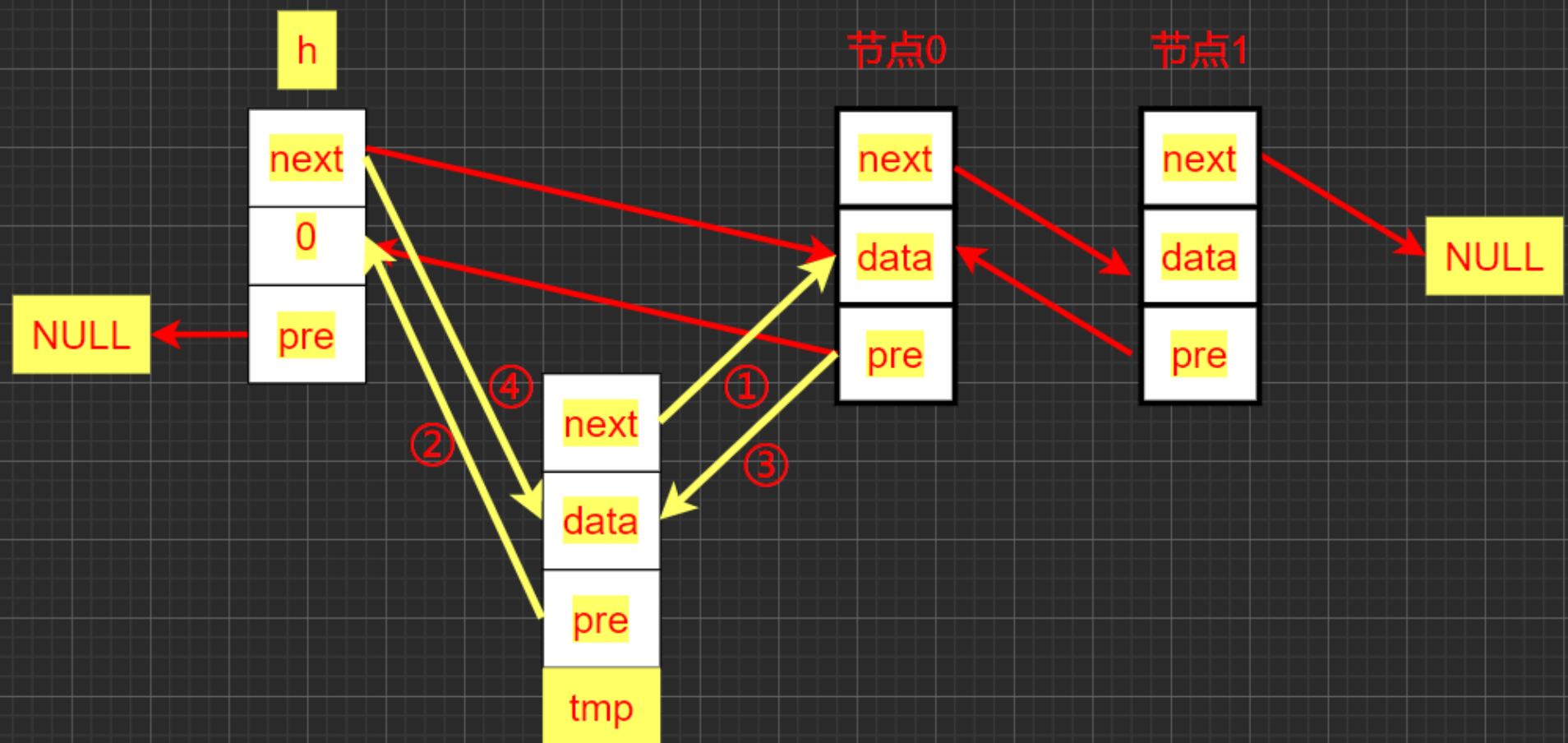
```
tmp->next = h->next;
tmp->pre = h;
h->next = tmp;
```



情况1：（双链表不为空链表）

```
tmp->next = h->next;
tmp->pre = h;
h->next->pre = tmp;
h->next = tmp;
```

双向链表

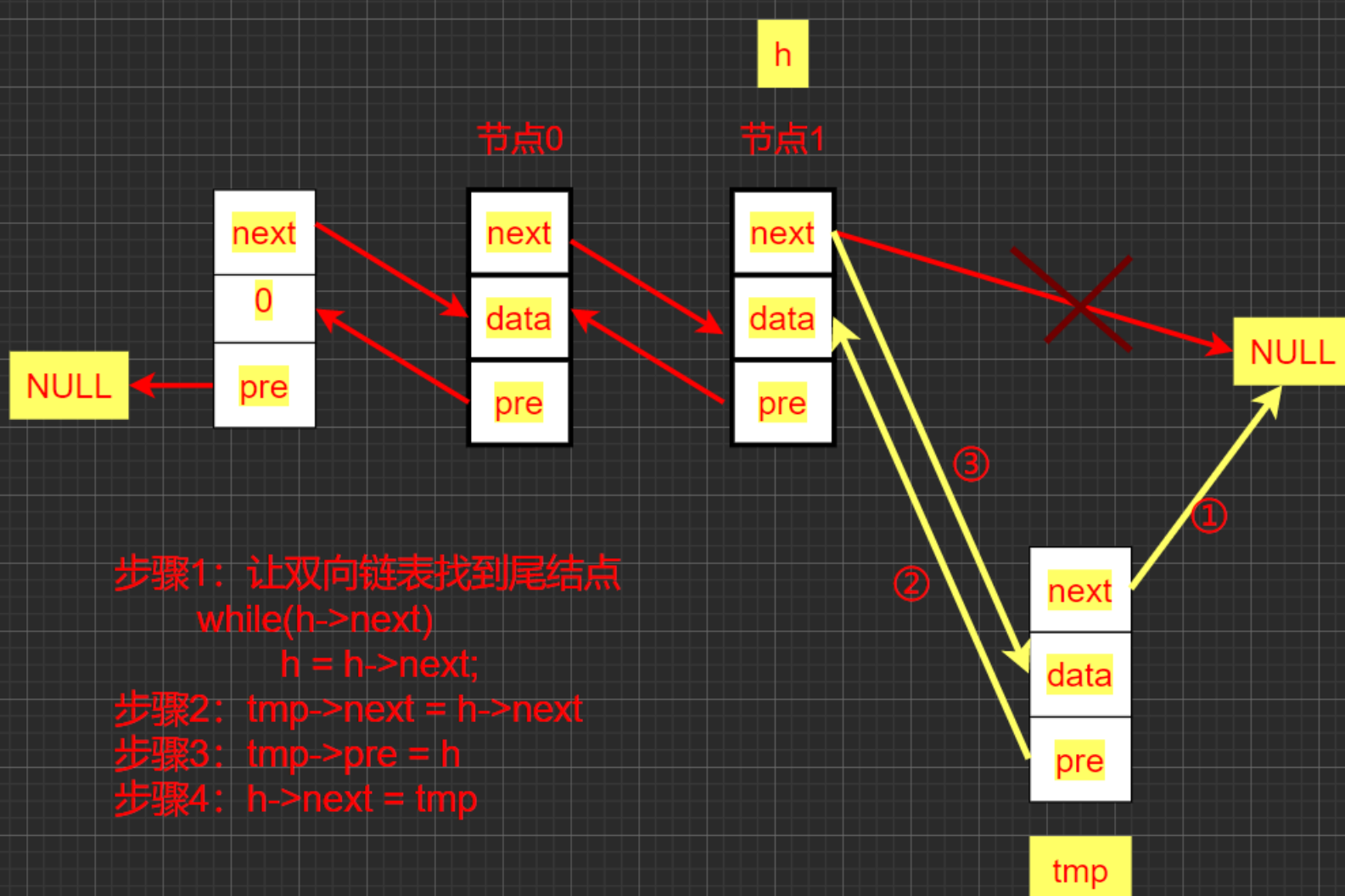


```

1 int DPListInsertHead(DPlist_t* h, datatype data)
2 {
3     DPlist_t* tmp;
4     tmp = (DPlist_t*)malloc(sizeof(*tmp));
5     if (tmp == NULL) {
6         printf("%s malloc memory error\n", __func__);
7         return -1;
8     }
9     tmp->data = data;
10
11     tmp->next = h->next;
12     tmp->pre = h;
13     if (h->next != NULL)
14         h->next->pre = tmp;
15     h->next = tmp;
16
17     return 0;
18 }

```

1.3.3双向链表尾插



```

1 int DPListInsertTail(DPlist_t* h, datatype data)
2 {
3     DPlist_t* tmp;

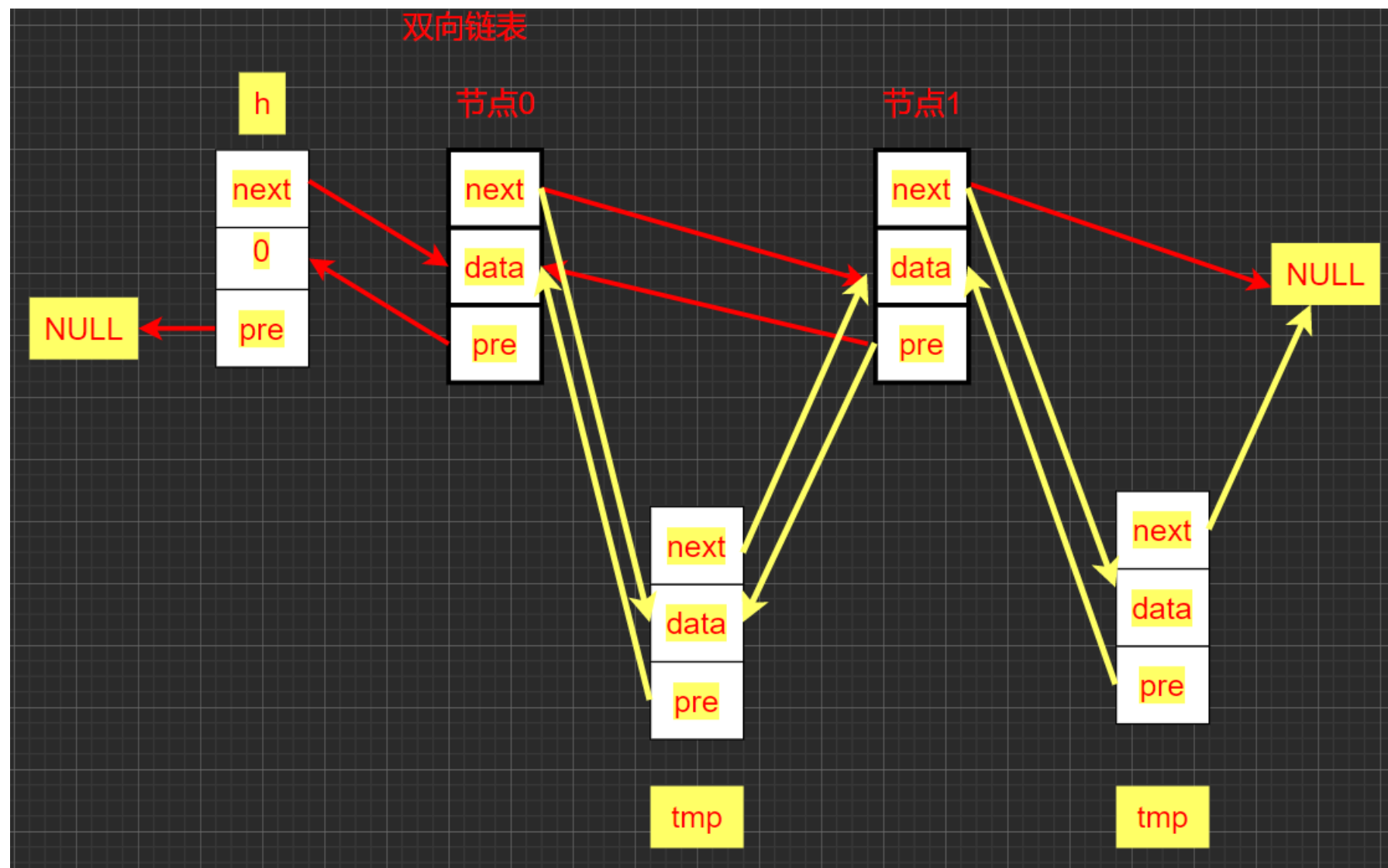
```

```

4 // 1.分配tmp节点, 将data存入
5 tmp = (DPlist_t*)malloc(sizeof(*tmp));
6 if (tmp == NULL) {
7     printf("%s malloc memory error\n", __func__);
8     return -1;
9 }
10 tmp->data = data;
11 // 2.让h走到尾节点
12 while (h->next)
13     h = h->next;
14 // 3.节点插入
15 tmp->next = h->next;
16 tmp->pre = h;
17 h->next = tmp;
18 return 0;
19 }

```

1.3.4双向链表位置插



```

1 int DPListInsertByPos(DPlist_t* h, int pos, datatype data)
2 {
3     if (pos < 0) {
4         printf("%s pos left error\n", __func__);
5         return -1;
6     }
7     while (h) {
8         if (pos != 0) {
9             h = h->next;
10            pos--;
11        } else {
12            // 找到插入的位置
13            DPlist_t* tmp;
14            tmp = (DPlist_t*)malloc(sizeof(*tmp));
15            if (tmp == NULL) {
16                printf("%s malloc memory error\n", __func__);
17                return -1;
18            }
19            tmp->data = data;
20
21            tmp->next = h->next;
22            tmp->pre = h;
23            if (h->next != NULL)
24                tmp->next->pre = tmp;
25            h->next = tmp;
26            return 0;
27        }
28    }
29    printf("%s pos right error\n", __func__);
30    return -1;
31 }

```

1.3.5双向链表的遍历

```
1 void DPListShow(DPlist_t* h)
2 {
3     printf("双向链表正向遍历: ");
4     while (h->next) {
5         printf("-%d", h->next->data);
6         h = h->next;
7     }
8     printf("-\n");
9     printf("双向链表逆向遍历: ");
10    while (h->pre) {
11        printf("-%d", h->data);
12        h = h->pre;
13    }
14    printf("-\n");
15 }
```

1.3.6双向链表判空

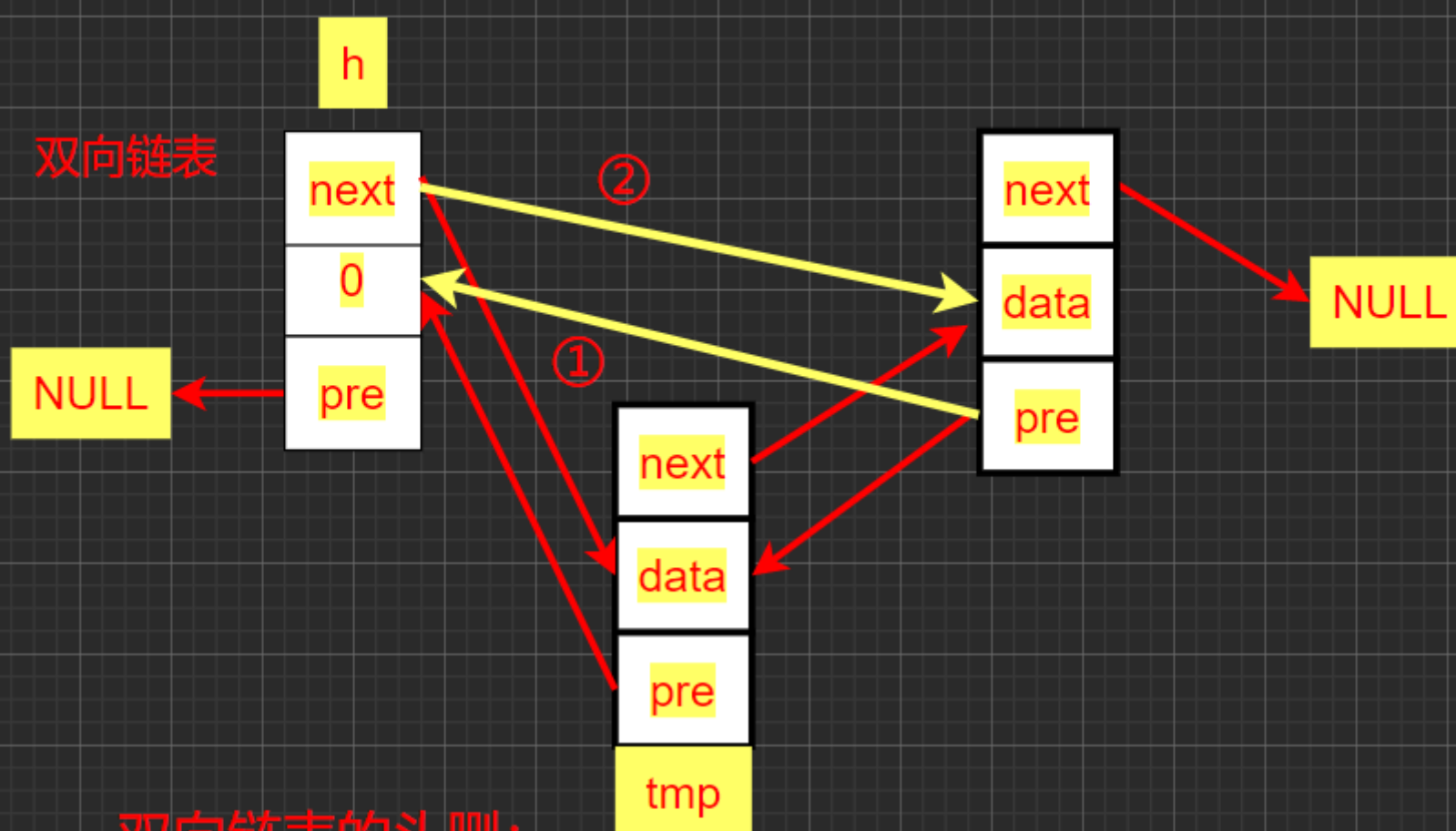
```
1 int DPListIsEmpty(DPlist_t* h)
2 {
3     return h->next == NULL ? 1 : 0;
4 }
```

1.3.7双向链表的头删

情况1：（双链表中只有一个节点）



情况2：（双链表中有多个节点）



双向链表的头删：

1.判断双向链表是否是空，如果是空退出，否则向下执行

2.让tmp记录被删除的节点 $tmp = h \rightarrow next$

3.删除的两条指针操作

$tmp \rightarrow next \rightarrow pre = h;$

$h \rightarrow next = tmp \rightarrow next;$

4.将tmp节点free

$free(tmp);$

$tmp = NULL;$

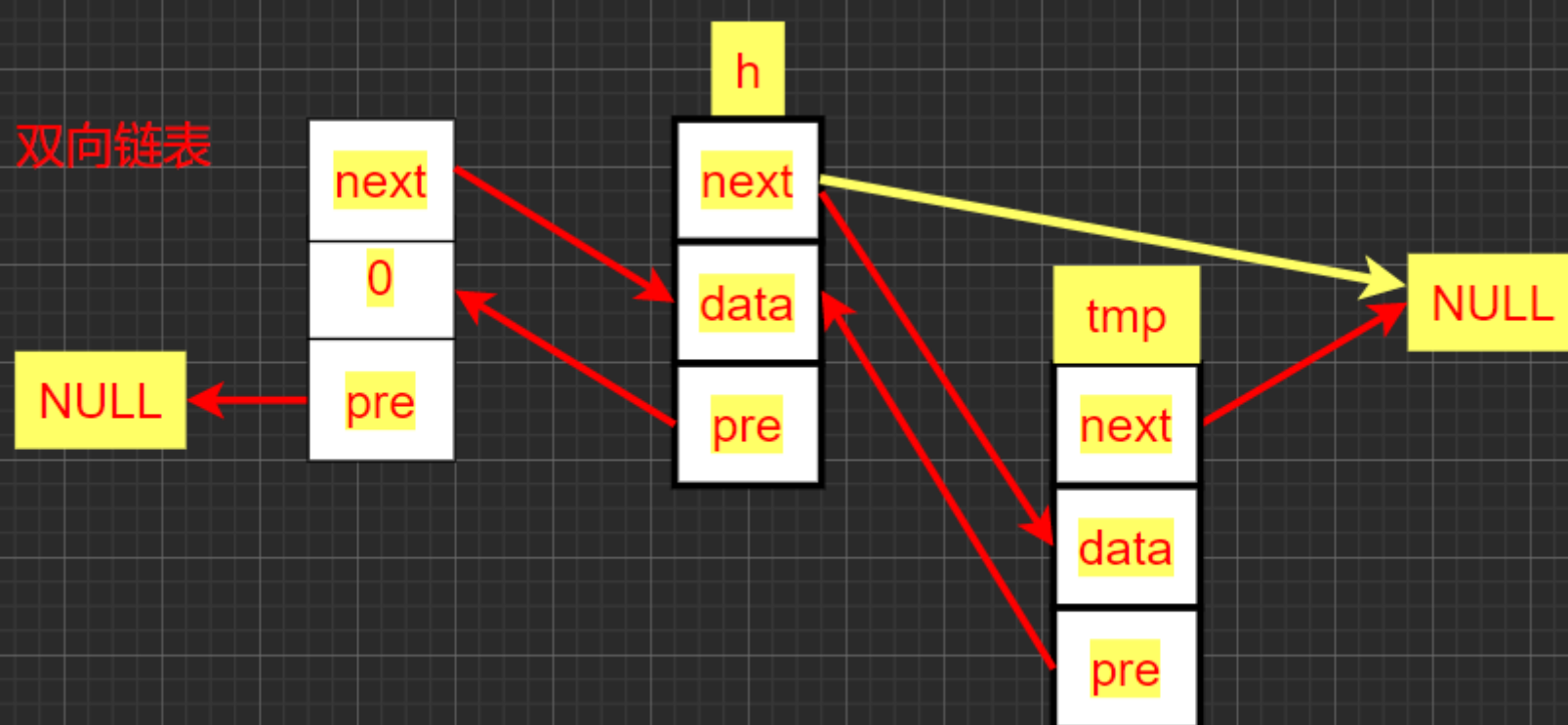
```

1 datatype DPListDeleteHead(DPlist_t* h)
2 {
3     datatype data;
4     DPlist_t* tmp;
5     if (DPListIsEmpty(h)) {
6         printf("%s list empty\n", __func__);
7         return (datatype)-1;
8     }
9
10    tmp = h->next;
11    if (tmp->next != NULL)
12        tmp->next->pre = h;
13    h->next = tmp->next;
14
15    data = tmp->data;
16    if (tmp != NULL) {
17        free(tmp);
18        tmp = NULL;
19    }
20    return data;
21 }

```

1.3.8双向链表的尾删

双向链表



双向链表的尾删：

1.判断双向链表是否时空，如果时空退出，否则向下执行

2.让h走到倒数第二个节点

```
while(h->nex->next){
    h = h->next;
}
```

3.让tmp记录被删除的节点 tmp = h->next

4.指针操作h->next = tmp->next;

5.释放tmp节点

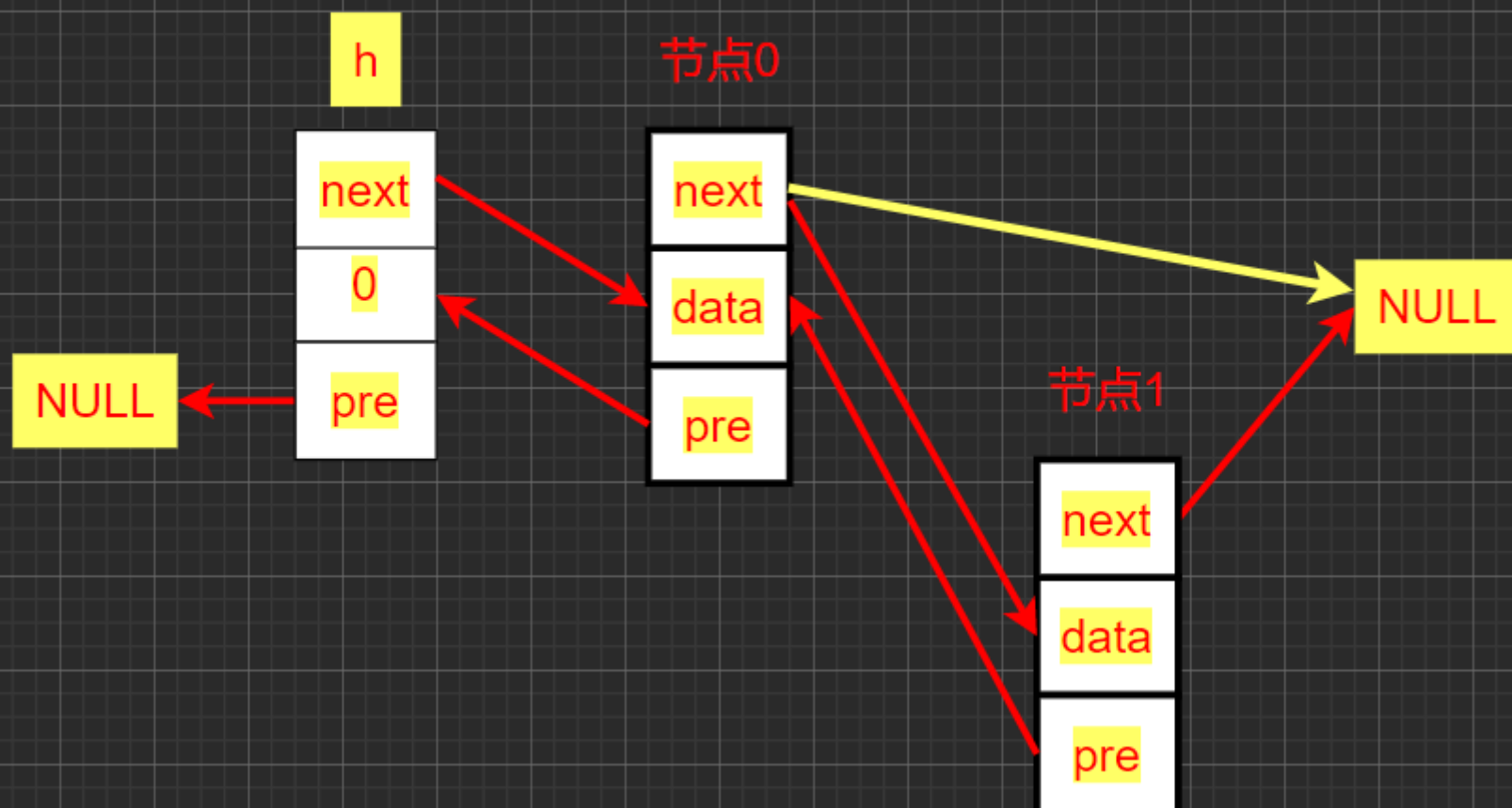
```
free(tmp);
tmp=NULL;
```

```

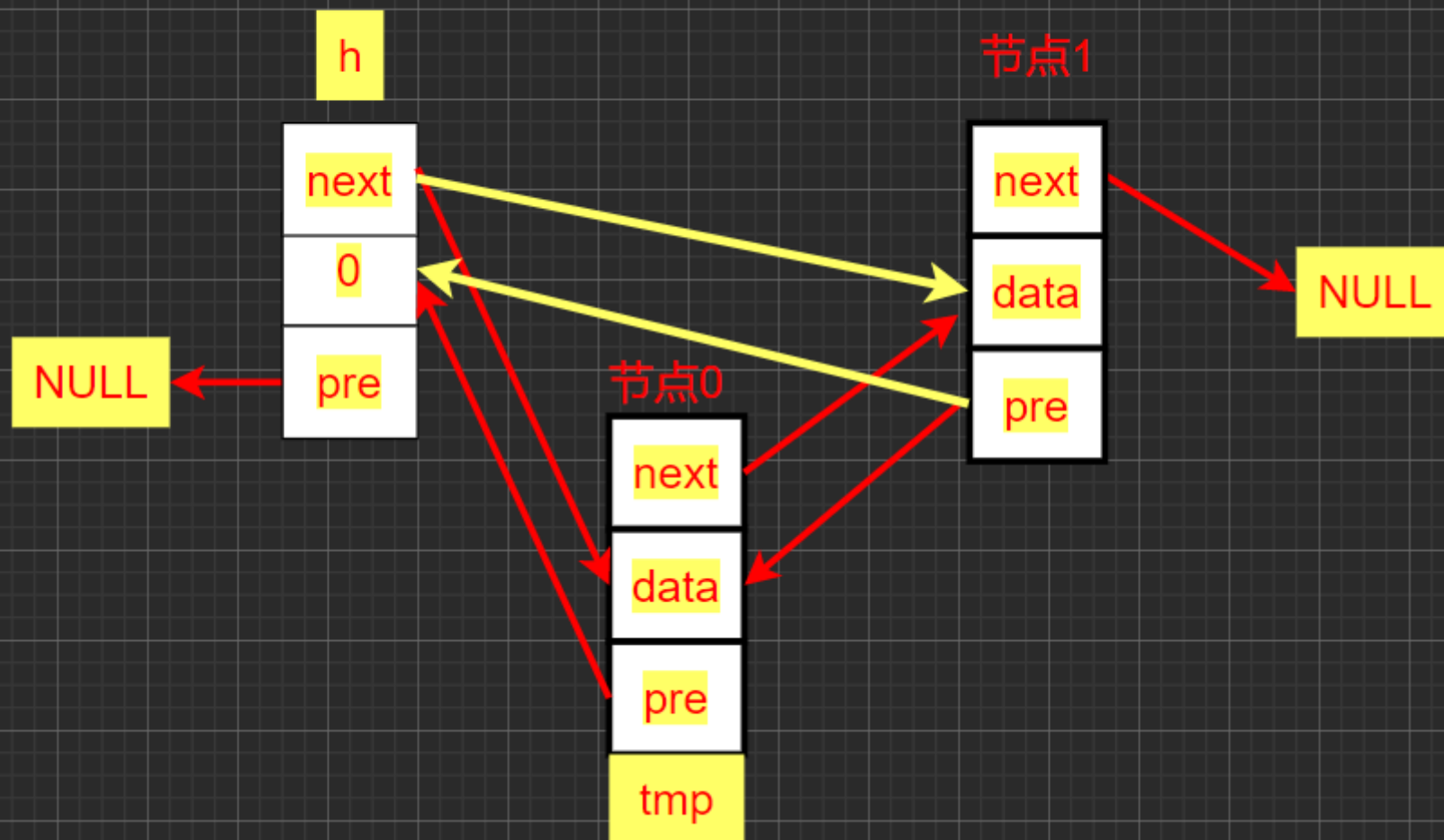
1 datatype DPListDeleteTail(DPList_t* h)
2 {
3     datatype data;
4     DPList_t* tmp;
5
6     if (DPListIsEmpty(h)) {
7         printf("%s list empty\n", __func__);
8         return (datatype)-1;
9     }
10
11     while (h->next->next)
12         h = h->next;
13
14     tmp = h->next;
15
16     h->next = tmp->next;
17     data = tmp->data;
18     if (tmp != NULL) {
19         free(tmp);
20         tmp = NULL;
21     }
22
23     return data;
24 }
```

1.3.9双向链表的位置删

双向链表



双向链表



```

1 datatype DPListDeleteByPos(DPList_t* h, int pos)
2 {
3     if (pos < 0) {
4         printf("%s pos left error\n", __func__);
5         return (datatype)-1;
6     }
7     while (h->next) {
8         if (pos != 0) {
9             h = h->next;
10            pos--;
11        } else {
12            // 找到删除的位置
13            DPList_t* tmp;
14            datatype data;
15            tmp = h->next;
16
17            if (tmp->next != NULL)
18                tmp->next->pre = h;
19            h->next = tmp->next;
20
21            data = tmp->data;
22            if (tmp != NULL) {
23                free(tmp);
24                tmp = NULL;
25            }
26        }
27    }
28 }

```



```
26         return data;
27     }
28 }
29 printf("%s pos right error\n", __func__);
30 return (datatype)-1;
31 }
```

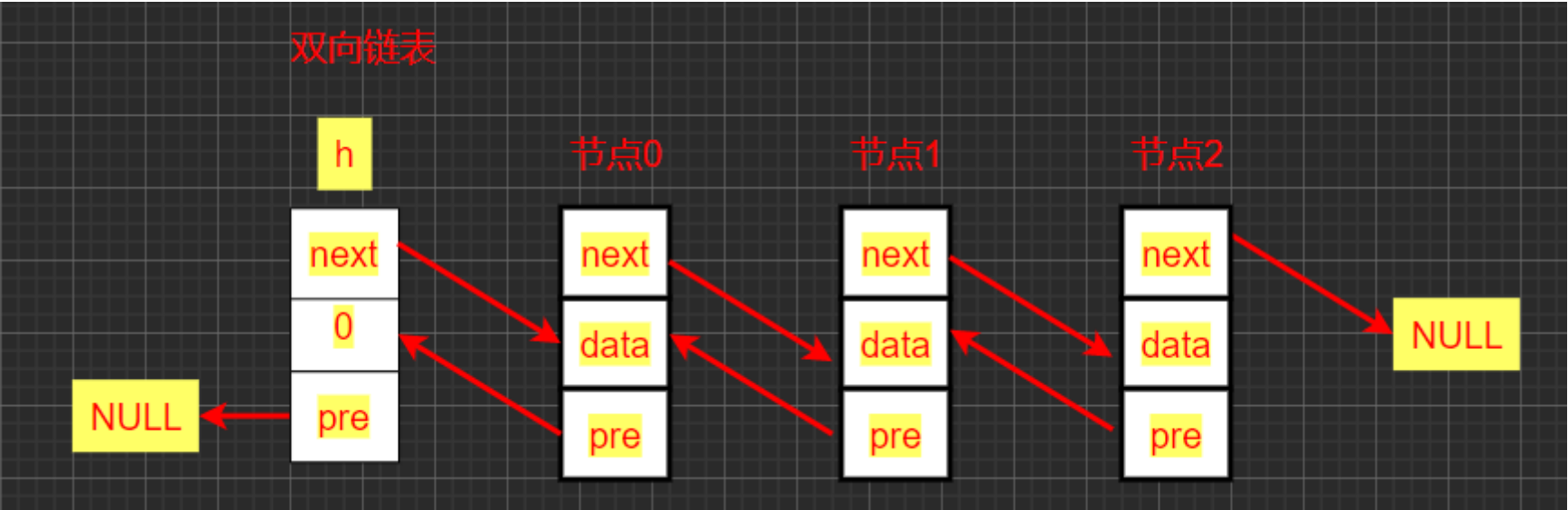
1.3.10双向链表的查询

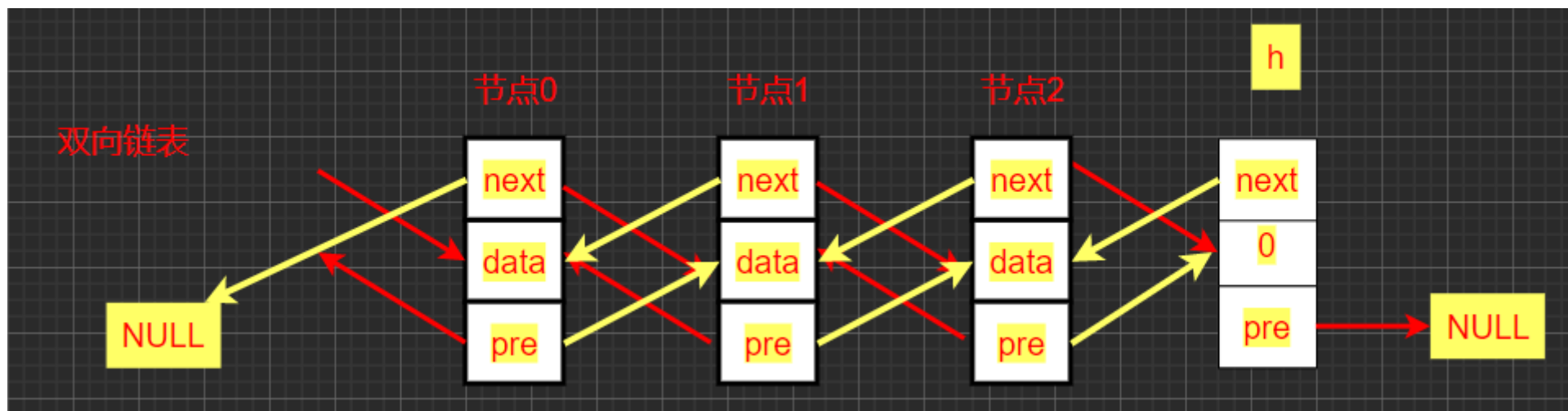
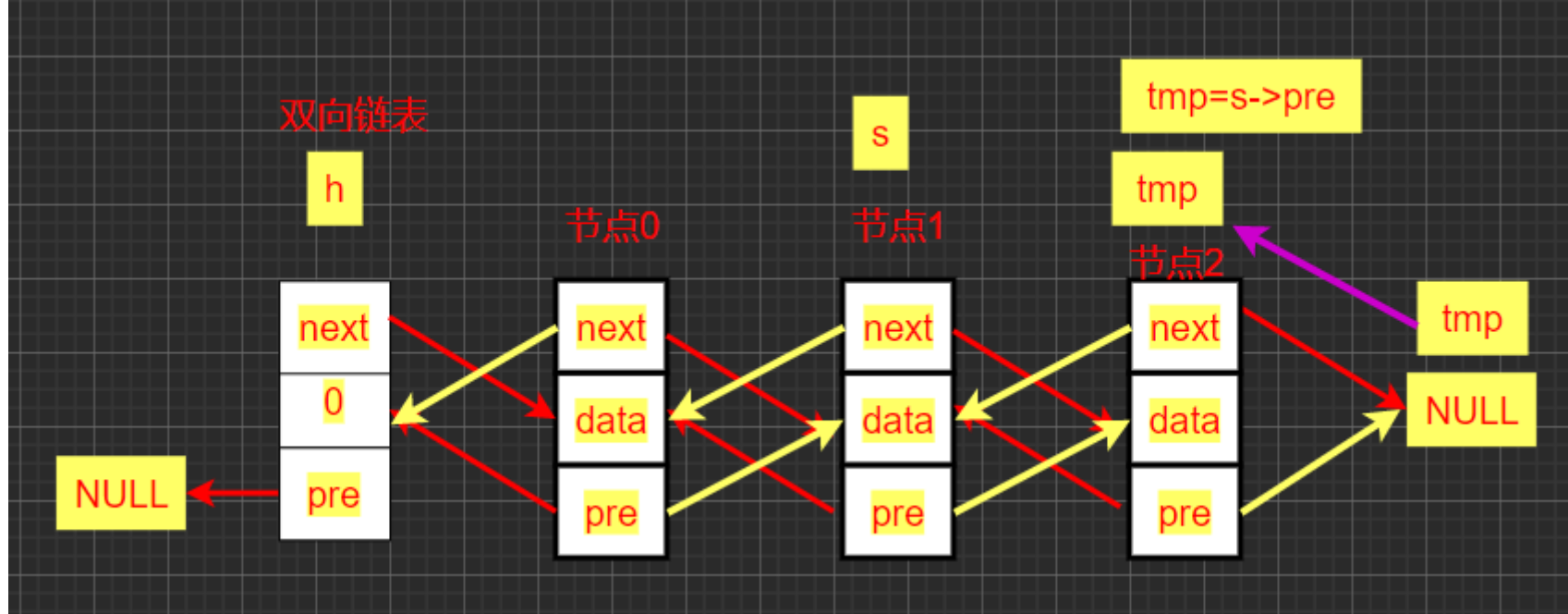
```
1 datatype DPListCheckDataByPos(DPlist_t* h, int pos)
2 {
3     if (pos < 0) {
4         printf("%s pos left error\n", __func__);
5         return (datatype)-1;
6     }
7     while (h->next) {
8         if (pos != 0) {
9             h = h->next;
10            pos--;
11        } else {
12            // 找到查询的位置
13            return h->next->data;
14        }
15    }
16    printf("%s pos right error\n", __func__);
17    return (datatype)-1;
18 }
```

1.3.11双向链表的更新

```
1 int DPListUpdateDataByPos(DPlist_t* h, int pos, datatype data)
2 {
3     if (pos < 0) {
4         printf("%s pos left error\n", __func__);
5         return -1;
6     }
7     while (h->next) {
8         if (pos != 0) {
9             h = h->next;
10            pos--;
11        } else {
12            // 找到更新的位置
13            h->next->data = data;
14            return 0;
15        }
16    }
17    printf("%s pos right error\n", __func__);
18    return -1;
19 }
```

1.3.12双向链表的逆序





```

1
2 void DPListReverse(DPList_t* h)
3 {
4     DPList_t *tmp, *s;
5     if (DPListIsEmpty(h))
6         return;
7     tmp = h->next;
8     while (tmp) {
9         s = tmp->pre;
10        tmp->pre = tmp->next;
11        tmp->next = s;
12        tmp = tmp->pre;
13    }
14    tmp = s->pre;
15    h->next->next = NULL;
16    h->next = tmp;
17    tmp->pre = h;
18 }

```

1.4整体代码

DPList.h

```

1 #ifndef __DPLIST_H__
2 #define __DPLIST_H__
3
4 #include <stdio.h>
5 #include <stdlib.h>
6
7 #define datatype int
8 typedef struct node {
9     datatype data;
10    struct node *pre, *next;
11 } DPList_t;
12 DPList_t* DPListCreate(void);
13 int DPListInsertHead(DPList_t* h, datatype data);
14 int DPListInsertTail(DPList_t* h, datatype data);
15 int DPListInsertByPos(DPList_t* h, int pos, datatype data);
16 int DPListIsEmpty(DPList_t* h);
17 void DPListShow(DPList_t* h);
18 datatype DPListDeleteHead(DPList_t* h);
19 datatype DPListDeleteTail(DPList_t* h);
20 datatype DPListDeleteByPos(DPList_t* h, int pos);
21 datatype DPListCheckDataByPos(DPList_t* h, int pos);
22 int DPListUpdateDataByPos(DPList_t* h, int pos, datatype data);
23 void DPListReverse(DPList_t* h);
24 #endif

```

DPlist.c

```
1  #include "DPlist.h"
2
3  DPlist_t* DPlistCreate(void)
4  {
5      DPlist_t* h;
6      h = (DPlist_t*)malloc(sizeof(*h));
7      if (h == NULL) {
8          printf("%s malloc memory error\n", __func__);
9          return NULL;
10     }
11     h->data = (datatype)0;
12     h->next = NULL;
13     h->pre = NULL;
14
15     return h;
16 }
17 int DPlistInsertHead(DPlist_t* h, datatype data)
18 {
19     DPlist_t* tmp;
20     tmp = (DPlist_t*)malloc(sizeof(*tmp));
21     if (tmp == NULL) {
22         printf("%s malloc memory error\n", __func__);
23         return -1;
24     }
25     tmp->data = data;
26
27     tmp->next = h->next;
28     tmp->pre = h;
29     if (h->next != NULL)
30         h->next->pre = tmp;
31     h->next = tmp;
32
33     return 0;
34 }
35 int DPlistInsertTail(DPlist_t* h, datatype data)
36 {
37     DPlist_t* tmp;
38     // 1.分配tmp节点, 将data存入
39     tmp = (DPlist_t*)malloc(sizeof(*tmp));
40     if (tmp == NULL) {
41         printf("%s malloc memory error\n", __func__);
42         return -1;
43     }
44     tmp->data = data;
45     // 2.让h走到尾节点
46     while (h->next)
47         h = h->next;
48     // 3.节点插入
49     tmp->next = h->next;
50     tmp->pre = h;
51     h->next = tmp;
52     return 0;
53 }
54 int DPlistInsertByPos(DPlist_t* h, int pos, datatype data)
55 {
56     if (pos < 0) {
57         printf("%s pos left error\n", __func__);
58         return -1;
59     }
60     while (h) {
61         if (pos != 0) {
62             h = h->next;
63             pos--;
64         } else {
65             // 找到插入的位置
66             DPlist_t* tmp;
67             tmp = (DPlist_t*)malloc(sizeof(*tmp));
68             if (tmp == NULL) {
69                 printf("%s malloc memory error\n", __func__);
70                 return -1;
71             }
72             tmp->data = data;
73
74             tmp->next = h->next;
75             tmp->pre = h;
76             if (h->next != NULL)
77                 tmp->next->pre = tmp;
78             h->next = tmp;
79             return 0;
80         }
81     }
82     printf("%s pos right error\n", __func__);
```

```
83     return -1;
84 }
85 void DPListShow(DPlist_t* h)
86 {
87     printf("双向链表正向遍历: ");
88     while (h->next) {
89         printf("-%d", h->next->data);
90         h = h->next;
91     }
92     printf("-\n");
93     printf("双向链表逆向遍历: ");
94     while (h->pre) {
95         printf("-%d", h->data);
96         h = h->pre;
97     }
98     printf("-\n");
99 }
100 int DPListIsEmpty(DPlist_t* h)
101 {
102     return h->next == NULL ? 1 : 0;
103 }
104 datatype DPListDeleteHead(DPlist_t* h)
105 {
106     datatype data;
107     DPlist_t* tmp;
108     if (DPListIsEmpty(h)) {
109         printf("%s list empty\n", __func__);
110         return (datatype)-1;
111     }
112
113     tmp = h->next;
114     if (tmp->next != NULL)
115         tmp->next->pre = h;
116     h->next = tmp->next;
117
118     data = tmp->data;
119     if (tmp != NULL) {
120         free(tmp);
121         tmp = NULL;
122     }
123
124     return data;
125 }
126 datatype DPListDeleteTail(DPlist_t* h)
127 {
128     datatype data;
129     DPlist_t* tmp;
130
131     if (DPListIsEmpty(h)) {
132         printf("%s list empty\n", __func__);
133         return (datatype)-1;
134     }
135
136     while (h->next->next)
137         h = h->next;
138
139     tmp = h->next;
140
141     h->next = tmp->next;
142     data = tmp->data;
143     if (tmp != NULL) {
144         free(tmp);
145         tmp = NULL;
146     }
147
148     return data;
149 }
150
151 datatype DPListDeleteByPos(DPlist_t* h, int pos)
152 {
153     if (pos < 0) {
154         printf("%s pos left error\n", __func__);
155         return (datatype)-1;
156     }
157     while (h->next) {
158         if (pos != 0) {
159             h = h->next;
160             pos--;
161         } else {
162             // 找到删除的位置
163             DPlist_t* tmp;
164             datatype data;
165             tmp = h->next;
166
```

```

167         if (tmp->next != NULL)
168             tmp->next->pre = h;
169         h->next = tmp->next;
170
171         data = tmp->data;
172         if (tmp != NULL) {
173             free(tmp);
174             tmp = NULL;
175         }
176         return data;
177     }
178 }
179 printf("%s pos right error\n", __func__);
180 return (datatype)-1;
181 }
182
183 datatype DPListCheckDataByPos(DPlist_t* h, int pos)
184 {
185     if (pos < 0) {
186         printf("%s pos left error\n", __func__);
187         return (datatype)-1;
188     }
189     while (h->next) {
190         if (pos != 0) {
191             h = h->next;
192             pos--;
193         } else {
194             // 找到查询的位置
195             return h->next->data;
196         }
197     }
198     printf("%s pos right error\n", __func__);
199     return (datatype)-1;
200 }
201 int DPListUpdateDataByPos(DPlist_t* h, int pos, datatype data)
202 {
203     if (pos < 0) {
204         printf("%s pos left error\n", __func__);
205         return -1;
206     }
207     while (h->next) {
208         if (pos != 0) {
209             h = h->next;
210             pos--;
211         } else {
212             // 找到更新的位置
213             h->next->data = data;
214             return 0;
215         }
216     }
217     printf("%s pos right error\n", __func__);
218     return -1;
219 }
220
221 void DPListReverse(DPlist_t* h)
222 {
223     DPlist_t *tmp, *s;
224     if (DPListIsEmpty(h))
225         return;
226     tmp = h->next;
227     while (tmp) {
228         s = tmp->pre;
229         tmp->pre = tmp->next;
230         tmp->next = s;
231         tmp = tmp->pre;
232     }
233     tmp = s->pre;
234
235     h->next->next = NULL;
236     h->next = tmp;
237     tmp->pre = h;
238 }

```

main.c

```

1 #include "DPList.h"
2 int main(int argc, const char* argv[])
3 {
4     DPlist_t* h;
5
6     h = DPListCreate();
7     if (h == NULL)
8         return -1;

```

```
9
10     DPListInsertHead(h, 123);
11     DPListInsertHead(h, 567);
12     DPListInsertHead(h, 888);
13     DPListInsertHead(h, 4390);
14     DPListShow(h);
15     // DPListInsertTail(h, 11);
16     // DPListInsertTail(h, 22);
17     // DPListShow(h);
18     // DPListInsertByPos(h, 1, 777);
19     // DPListShow(h);
20
21     // DPListDeleteHead(h);
22     // DPListShow(h);
23     // DPListDeleteHead(h);
24     // DPListDeleteTail(h);
25     // DPListShow(h);
26     // DPListDeleteByPos(h,1);
27     // DPListShow(h);
28
29     // printf("check data = %d\n",DPListCheckDataByPos(h,1));
30     // DPListUpdateDataByPos(h,1,666);
31     // DPListShow(h);
32     puts("-----");
33     DPListReverse(h);
34     DPListShow(h);
35     return 0;
36 }
```