

- 1.单链表
- 1.1什么是单链表

1.2单链表的结构

1.3单链表的常见操作

1.3.1创建单链表

1.3.2单链表的头插

1.3.3单链表的尾插

1.3.4单链表任意位置插入节点

1.3.5单链表的遍历

1.3.6单链表的判空

1.3.7单链表头删

1.3.8单链表尾删

1.3.9单链表任意位置删除

1.3.10单链表根据位置查询数据

1.3.11单链表根据位置更新数据

1.3.12单链表逆序

1.3.13单链表排序(直接插入排序)

1.4单链表整体代码

linklist.h

linklist.c

main.c
- 2.单向循环链表

2.1单向循环链表的特点

2.2单向循环链表结构

2.3单向循环链表的常见操作

2.3.1创建单向循环链表

2.3.2单向循环链表的头插

2.3.3单向循环链表的遍历

2.3.4单向循环链表判空

2.3.5单向循环链表头删

2.3.6单向循环链根据位置查询数据

2.3.7单向循环链根据位置更新数据

2.4单向循环链表整体代码

looplist.h

looplist.c

main.c

3.作业

1.单链表

1.1什么是单链表

单链表：它是线性表的链式存储。

1.2单链表的结构

1.节点组成

A diagram showing a single node structure. It consists of two adjacent white rectangular boxes on a dark grid background. The left box contains the word 'data' in red text, and the right box contains the word 'next' in red text. Both words are highlighted with yellow rectangular backgrounds.

data:节点的数据域

next:节点的指针域

2.单链表节点结构体封装

```
1 | #define datatype int
2 | typedef struct node{
3 |     datatype data;      //数据域
4 |     struct node *next;  //指向下一个节点的指针域
5 | }linklist_t;
```

3.链表组成

A diagram illustrating the composition of a linked list. At the top left, a yellow box labeled 'h' (head) points to the first node. The first node is a white box divided into '0' and 'next' fields. The 'next' field points to a second node, which is a white box divided into 'data' and 'next' fields. This second node's 'next' field points to a third node, also a white box with 'data' and 'next' fields. The third node's 'next' field points to a yellow box labeled 'Tail'. The 'Tail' box points to a final white box divided into 'data' and 'next' fields. The final node's 'next' field points to a yellow box labeled 'NULL'. All pointers are represented by red arrows.

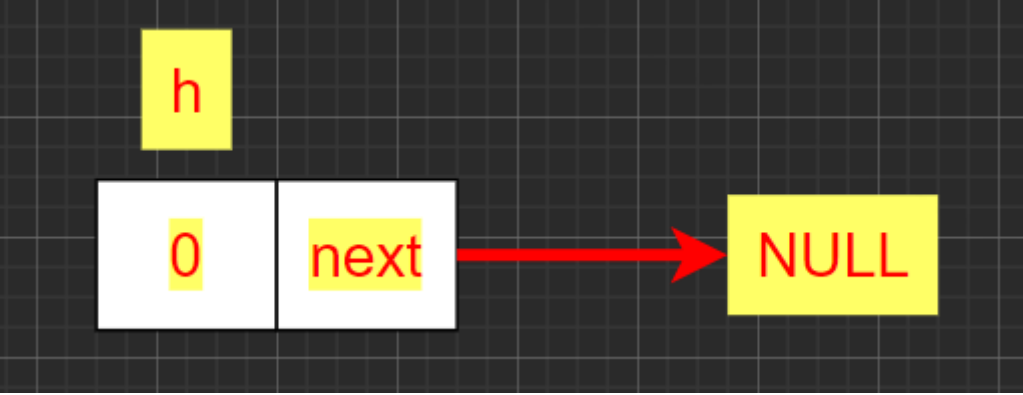
h: 单链表的头节点，只有后继节点，没有前驱节点，h->data存放的是0(不存数据)

Tail:它是单链表的尾节点，尾结点只有前驱，没有后继，Tail->next指针域指向的NULL

取余的节点都是中间节点，它们即有前驱节点又有后继节点。

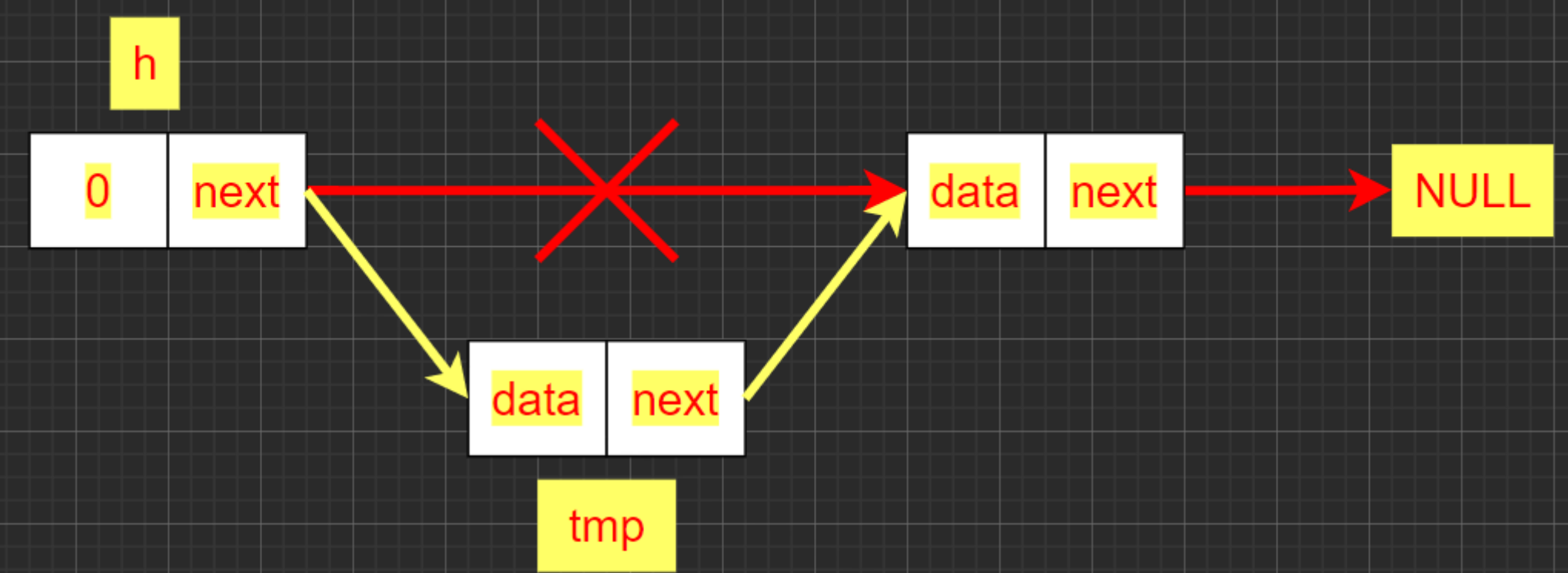
1.3单链表的常见操作

1.3.1创建单链表



```
1 linklist_t* LinkListCreate(void)
2 {
3     linklist_t* h = NULL;
4     // 1.分配节点的内存
5     h = (linklist_t*)malloc(sizeof(*h));
6     if (h == NULL) {
7         printf("%s malloc memory error\n", __func__);
8         return NULL;
9     }
10    // 2.对节点中的成员初始化
11    h->data = (datatype)0;
12    h->next = NULL;
13
14    // 3.将节点的地址返回
15    return h;
16 }
```

1.3.2单链表的头插

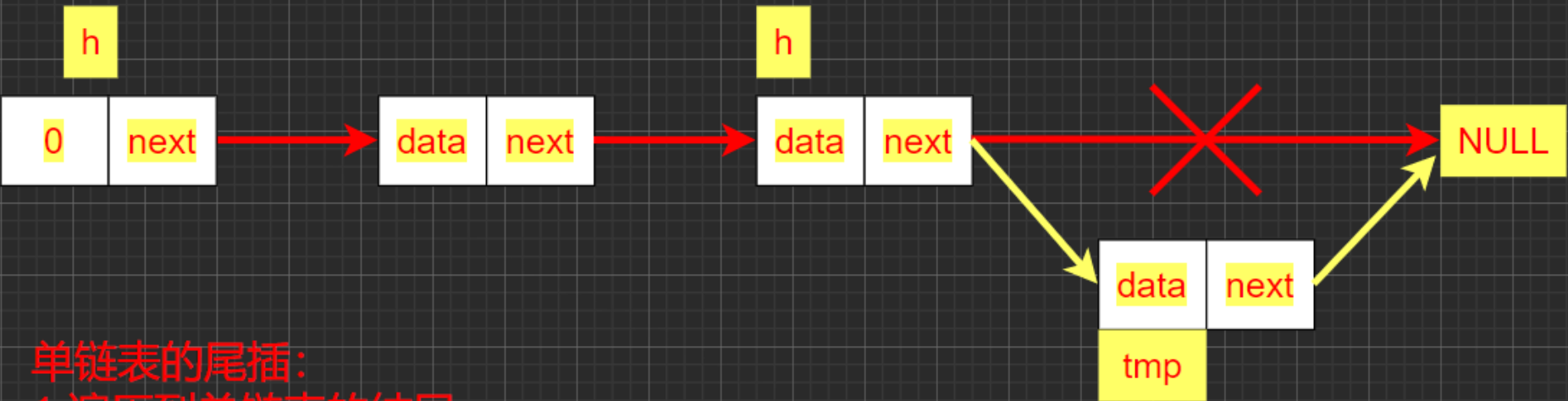


- 单链表的头插：
- 1.先分配tmp节点
 - 2.将tmp节点中的数据存放进来tmp->data = data
 - 3.将h->next赋值给tmp->next
 - 4.将tmp赋值给h->next

```
1 int LinkListInsertHead(linklist_t *h,datatype data)
2 {
3     linklist_t *tmp;
4     // 1.分配tmp节点
5     tmp = (linklist_t *)malloc(sizeof(*tmp));
6     if(tmp == NULL){
7         printf("%s malloc memory error\n",__func__);
8         return -1;
9     }
10    // 2.将数据存入
```

```
11 tmp->data = data;
12 // 3.将tmp节点头插到链表中
13 tmp->next = h->next;
14 h->next = tmp;
15
16 return 0;
17 }
```

1.3.3单链表的尾插

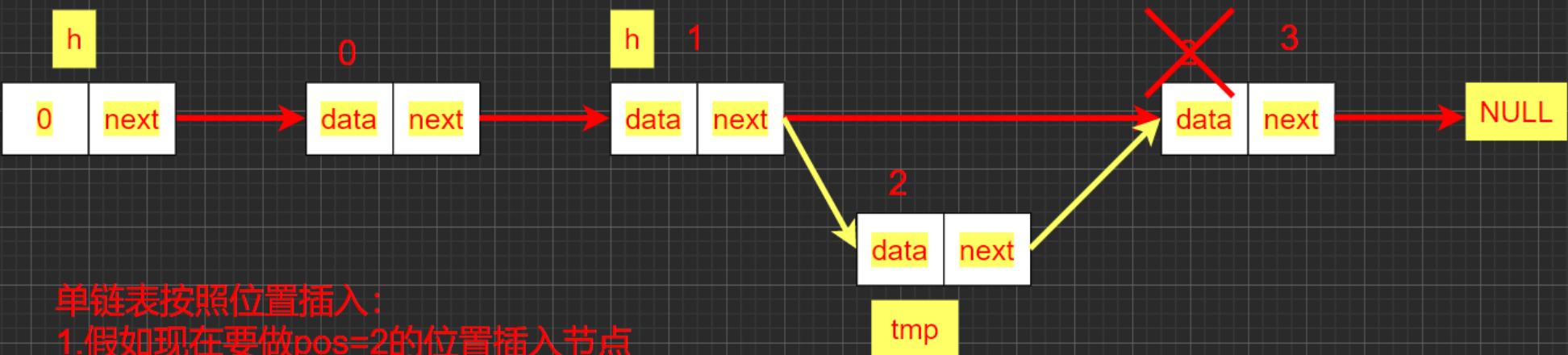


单链表的尾插：

- 1.遍历到单链表的结尾
- 2.分配tmp节点，将数据存放到tmp->data中
- 3.将tmp节点插入单链表
 `tmp->next = h->next;`
 `h->next = tmp;`

```
1 int LinkedListInsertTail(linklist_t* h, datatype data)
2 {
3     linklist_t* tmp;
4     // 1.遍历到链表的结尾
5     while (h->next)
6         h = h->next;
7     // 2.分配tmp节点内存，将data存放进来
8     tmp = (linklist_t*)malloc(sizeof(*tmp));
9     if (tmp == NULL) {
10         printf("%s malloc memory error\n", __func__);
11         return -1;
12     }
13     tmp->data = data;
14     // 3.将tmp节点插入单链表
15     tmp->next = h->next;
16     h->next = tmp;
17
18     return 0;
19 }
```

1.3.4单链表任意位置插入节点



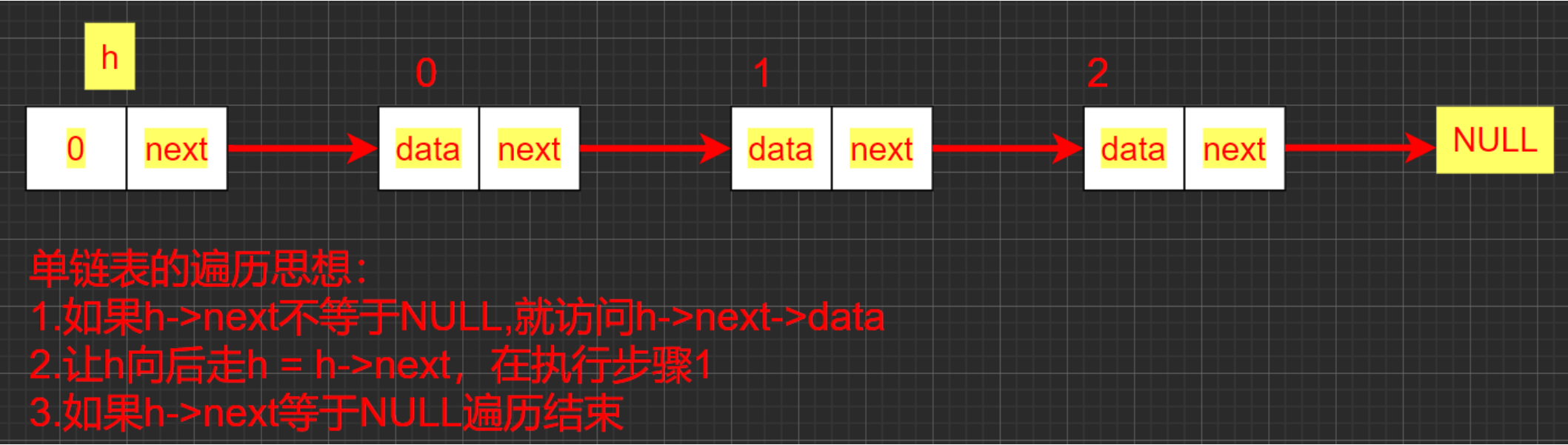
单链表按照位置插入：

- 1.假如现在要做pos=2的位置插入节点
- 2.判断pos是否是0，如果pos不是0，
 `h = h->next`
 `pos--;`
- 3.循环执行步骤2，直到pos=0退出
- 4.分配tmp节点，将数据存入
- 5.节点插入单链表
 `tmp->next = h->next`
 `h->next = tmp`

```
1 int LinkedListInsertByPos(linklist_t* h, int pos, datatype data)
```

```
2 {
3     linklist_t* tmp;
4     // 1.判断位置（左侧）合法性
5     if (pos < 0) {
6         printf("%s pos left error\n", __func__);
7         return -1;
8     }
9     // 2.如果pos不为0，让h向后走，然后pos--,(退出条件是链表到结尾)
10    while (h) { //这里写h的原因是想让h多走一次，因为节点可以在尾部位置插入
11        if (pos != 0) {
12            h = h->next;
13            pos--;
14        } else {
15            // 3.分配tmp节点，将数据存入
16            tmp = (linklist_t*)malloc(sizeof(*tmp));
17            if (tmp == NULL) {
18                printf("%s malloc memory error\n", __func__);
19                return -1;
20            }
21            tmp->data = data;
22            // 4.将tmp节点插入单链表
23            tmp->next = h->next;
24            h->next = tmp;
25            return 0;
26        }
27    }
28
29    printf("%s pos right error\n", __func__);
30    return -1;
31 }
```

1.3.5单链表的遍历



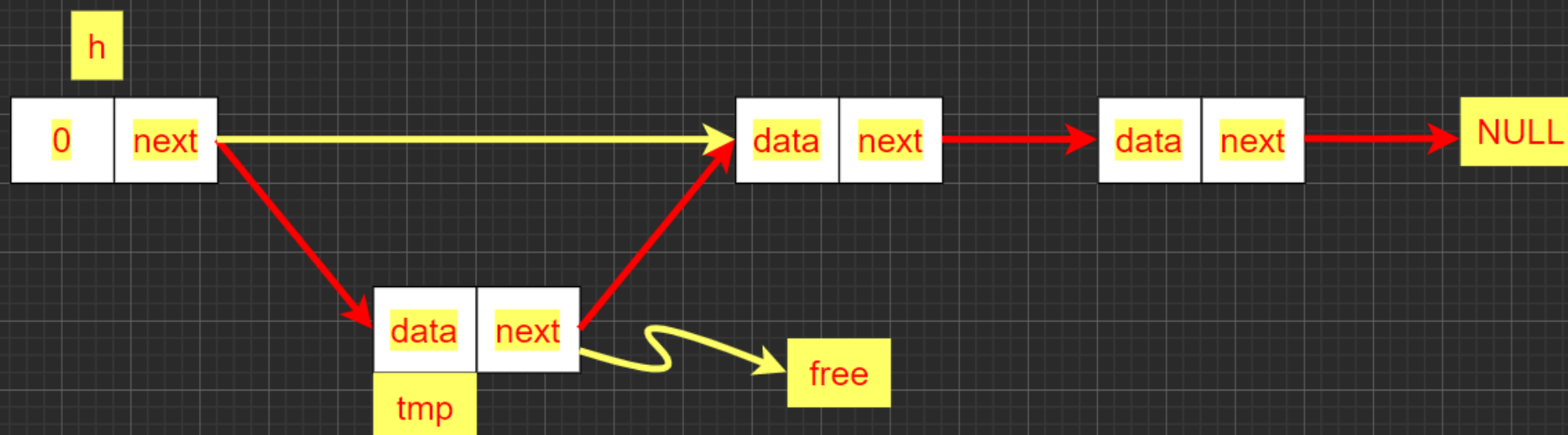
```
1 void LinkListShow(linklist_t* h)
2 {
3     while(h->next){
4         printf("-%d",h->next->data);
5         h = h->next;
6     }
7     printf("-\n");
8 }
```

1.3.6单链表的判空

判断h->next是否是NULL，如果是NULL就说明单链表是空的

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

1.3.7单链表头删



单链表头删除:

1.判断单链表是否是空，空直接退出，否则执行后面步骤

2.让tmp指针记录被删除的节点

tmp = h->next;

3.让h->next指向tmp->next

h->next = tmp->next;

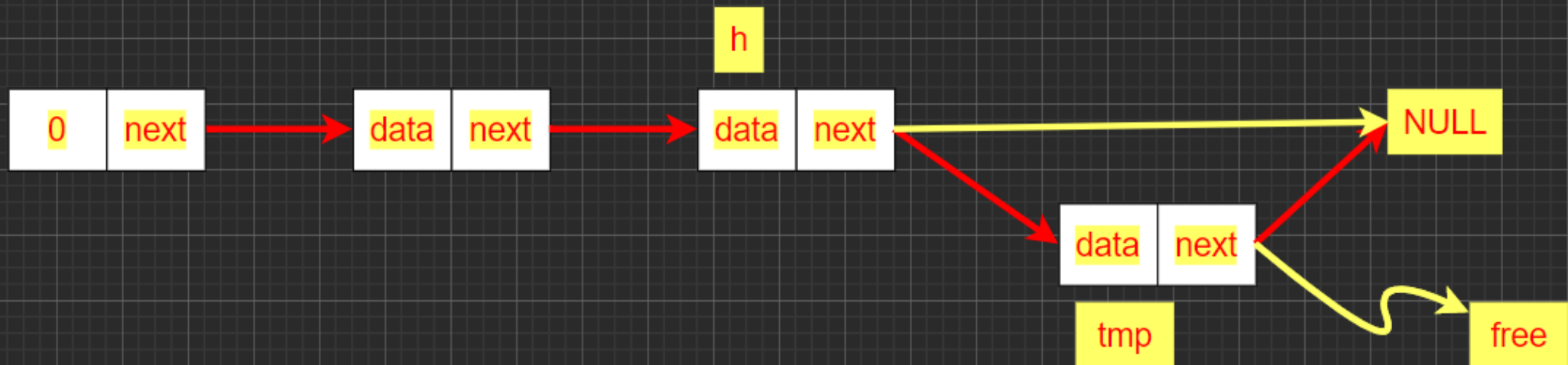
4.将tmp节点的内存释放掉

free(tmp);

tmp = NULL;

```
1 datatype LinkedListDeleteHead(linklist_t* h)
2 {
3     datatype data;
4     linklist_t* tmp;
5     // 1.判空
6     if (LinkedListIsEmpty(h)) {
7         printf("%s is empty\n", __func__);
8         return (datatype)-1;
9     }
10    // 2.让tmp指向h->next节点
11    tmp = h->next;
12    // 3.让h->next指向tmp->next
13    h->next = tmp->next;
14    // 4.释放节点内存
15    data = tmp->data;
16    if (tmp != NULL) {
17        free(tmp);
18        tmp = NULL;
19    }
20    return data;
21 }
```

1.3.8单链表尾删



单链表的尾删：

1.判断单链表是否为空，如果为空退出，如果不为空执行后面步骤

2.让h走到倒数第二个节点位置

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

3.让tmp记录h->next节点

4.h->next = tmp->next

5.释放tmp节点的内存

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

```
1 datatype LinkedListDeleteTail(linklist_t* h)
2 {
3     datatype data;
4     linklist_t* tmp;
5     // 1.判空
6     if (LinkedListIsEmpty(h)) {
7         printf("%s is empty\n", __func__);
8         return (datatype)-1;
9     }
10    // 2.让h走到倒数第二个节点停下
11    while(h->next->next)
12        h = h->next;
13    // 3.让tmp指向h->next节点
14    tmp = h->next;
15    // 4.让h->next指向tmp->next
16    h->next = tmp->next;
17    // 5.释放节点内存
18    data = tmp->data;
19    if (tmp != NULL) {
20        free(tmp);
21        tmp = NULL;
22    }
23    return data;
24 }
```

问：如果有一条单链表，有一个指针p指向这条单链表的其中一个节点，这个节点的下一个节点不为NULL，如何删除当前节点？

答：先将后面节点的内容覆盖到当前节点，然后将p指向的下一个节点的内存free掉即可。

1.3.9单链表任意位置删除

```
1 datatype LinkedListDeleteByPos(linklist_t* h, int pos)
2 {
3     linklist_t* tmp;
4     datatype data;
5     // 1.判断位置（左侧）合法性
6     if (pos < 0) {
7         printf("%s pos left error\n", __func__);
8         return -1;
9     }
10    // 2.如果pos不为0，让h向后走，然后pos--，(退出条件是链表到结尾)
11    while (h->next) {
12        if (pos != 0) {
13            h = h->next;
14            pos--;
15        } else {
16            // 3.让tmp指向h->next节点
17            tmp = h->next;
18            // 4.让h->next指向tmp->next
19            h->next = tmp->next;
20            // 5.释放节点内存
```

```
21         data = tmp->data;
22         if (tmp != NULL) {
23             free(tmp);
24             tmp = NULL;
25         }
26         return data;
27     }
28 }
29
30 printf("%s pos right error\n", __func__);
31 return -1;
32 }
```

1.3.10单链表根据位置查询数据

```
1 datatype LinkedListCheckDataByPos(linklist_t* h, int pos)
2 {
3     // 1.判断位置（左侧）合法性
4     if (pos < 0) {
5         printf("%s pos left error\n", __func__);
6         return (datatype)-1;
7     }
8     // 2.如果pos不为0, 让h向后走, 然后pos--, (退出条件是链表到结尾)
9     while (h->next) {
10         if (pos != 0) {
11             h = h->next;
12             pos--;
13         } else {
14             return h->next->data;
15         }
16     }
17
18     printf("%s pos right error\n", __func__);
19     return (datatype)-1;
20 }
```

1.3.11单链表根据位置更新数据

```
1 int LinkedListUpdateDataByPos(linklist_t* h, int pos, datatype data)
2 {
3     // 1.判断位置（左侧）合法性
4     if (pos < 0) {
5         printf("%s pos left error\n", __func__);
6         return -1;
7     }
8     // 2.如果pos不为0, 让h向后走, 然后pos--, (退出条件是链表到结尾)
9     while (h->next) {
10         if (pos != 0) {
11             h = h->next;
12             pos--;
13         } else {
14             h->next->data = data;
15             return 0;
16         }
17     }
18
19     printf("%s pos right error\n", __func__);
20     return -1;
21 }
```

1.3.12单链表逆序

h-10-30-20-40

h-40-20-30-10

```
1 void LinkedListReverse(linklist_t *h)
2 {
3     linklist_t *tmp,*s;
4     tmp = h->next;
5     h->next = NULL;
6     while(tmp){
7         s = tmp;
8         tmp = tmp->next;
9         s->next = h->next;
10        h->next = s;
11    }
12 }
```


1.3.13单链表排序(直接插入排序)

tmp-5-30-20-40

h-10-NULL

```
1 void LinkListInsertSort(linklist_t* h)
2 {
3     linklist_t *tmp, *s, *th = h;
4     tmp = h->next;
5     h->next = NULL;
6     while (tmp) {
7         s = tmp;
8         tmp = tmp->next;
9         while (h->next != NULL && h->next->data < s->data) {
10             h = h->next;
11         }
12         s->next = h->next;
13         h->next = s;
14         h = th;
15     }
16 }
```

1.4单链表整体代码

linklist.h

```
1 #ifndef __LINKLIST_H__
2 #define __LINKLIST_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* next;
11 } linklist_t;
12
13 linklist_t* LinkListCreate(void);
14 int LinkListInsertHead(linklist_t* h, datatype data);
15 int LinkListInsertTail(linklist_t* h, datatype data);
16 int LinkListInsertByPos(linklist_t* h, int pos, datatype data);
17 void LinkListShow(linklist_t* h);
18 int LinkListIsEmpty(linklist_t *h);
19 datatype LinkListDeleteHead(linklist_t *h);
20 datatype LinkListDeleteTail(linklist_t *h);
21 datatype LinkListDeleteByPos(linklist_t *h,int pos);
22 datatype LinkListCheckDataByPos(linklist_t *h,int pos);
23 int LinkListUpdateDataByPos(linklist_t *h,int pos,datatype data);
24 void LinkListReverse(linklist_t *h);
25 void LinkListInsertSort(linklist_t *h);
26 #endif
```

linklist.c

```
1 #include "linklist.h"
2
3 linklist_t* LinkListCreate(void)
4 {
5     linklist_t* h;
6     // 1.分配节点的内存
7     h = (linklist_t*)malloc(sizeof(*h));
8     if (h == NULL) {
9         printf("%s malloc memory error\n", __func__);
10        return NULL;
11    }
12    // 2.对节点中的成员初始化
13    h->data = (datatype)0;
14    h->next = NULL;
15
16    // 3.将节点的地址返回
17    return h;
18 }
19
20 int LinkListInsertHead(linklist_t* h, datatype data)
21 {
22     linklist_t* tmp;
23     // 1.分配tmp节点
24     tmp = (linklist_t*)malloc(sizeof(*tmp));
25     if (tmp == NULL) {
26         printf("%s malloc memory error\n", __func__);
```



```
27     return -1;
28 }
29 // 2.将数据存入
30 tmp->data = data;
31 // 3.将tmp节点头插到链表中
32 tmp->next = h->next;
33 h->next = tmp;
34
35 return 0;
36 }
37 int LinkedListInsertTail(linklist_t* h, datatype data)
38 {
39     linklist_t* tmp;
40     // 1.遍历到链表的结尾
41     while (h->next)
42         h = h->next;
43     // 2.分配tmp节点内存, 将data存放进来
44     tmp = (linklist_t*)malloc(sizeof(*tmp));
45     if (tmp == NULL) {
46         printf("%s malloc memory error\n", __func__);
47         return -1;
48     }
49     tmp->data = data;
50     // 3.将tmp节点插入单链表
51     tmp->next = h->next;
52     h->next = tmp;
53
54     return 0;
55 }
56
57 int LinkedListInsertByPos(linklist_t* h, int pos, datatype data)
58 {
59     linklist_t* tmp;
60     // 1.判断位置 (左侧) 合法性
61     if (pos < 0) {
62         printf("%s pos left error\n", __func__);
63         return -1;
64     }
65     // 2.如果pos不为0, 让h向后走, 然后pos--, (退出条件是链表到结尾)
66     while (h) {
67         if (pos != 0) {
68             h = h->next;
69             pos--;
70         } else {
71             // 3.分配tmp节点, 将数据存入
72             tmp = (linklist_t*)malloc(sizeof(*tmp));
73             if (tmp == NULL) {
74                 printf("%s malloc memory error\n", __func__);
75                 return -1;
76             }
77             tmp->data = data;
78             // 4.将tmp节点插入单链表
79             tmp->next = h->next;
80             h->next = tmp;
81             return 0;
82         }
83     }
84
85     printf("%s pos right error\n", __func__);
86     return -1;
87 }
88
89 void LinkedListShow(linklist_t* h)
90 {
91     while (h->next) {
92         printf("-%d", h->next->data);
93         h = h->next;
94     }
95     printf("-\n");
96 }
97 int LinkedListIsEmpty(linklist_t* h)
98 {
99     return (h->next == NULL) ? 1 : 0;
100 }
101 datatype LinkedListDeleteHead(linklist_t* h)
102 {
103     datatype data;
104     linklist_t* tmp;
105     // 1.判空
106     if (LinkedListIsEmpty(h)) {
107         printf("%s is empty\n", __func__);
108         return (datatype)-1;
109     }
110     // 2.让tmp指向h->next节点
```

```
111     tmp = h->next;
112     // 3.让h->next指向tmp->next
113     h->next = tmp->next;
114     // 4.释放节点内存
115     data = tmp->data;
116     if (tmp != NULL) {
117         free(tmp);
118         tmp = NULL;
119     }
120
121     return data;
122 }
123
124 datatype LinkListDeleteTail(linklist_t* h)
125 {
126     datatype data;
127     linklist_t* tmp;
128     // 1.判空
129     if (LinkListIsEmpty(h)) {
130         printf("%s is empty\n", __func__);
131         return (datatype)-1;
132     }
133     // 2.让h走到倒数第二个节点停下
134     while (h->next->next)
135         h = h->next;
136     // 3.让tmp指向h->next节点
137     tmp = h->next;
138     // 4.让h->next指向tmp->next
139     h->next = tmp->next;
140     // 5.释放节点内存
141     data = tmp->data;
142     if (tmp != NULL) {
143         free(tmp);
144         tmp = NULL;
145     }
146     return data;
147 }
148
149 datatype LinkListDeleteByPos(linklist_t* h, int pos)
150 {
151     linklist_t* tmp;
152     datatype data;
153     // 1.判断位置（左侧）合法性
154     if (pos < 0) {
155         printf("%s pos left error\n", __func__);
156         return (datatype)-1;
157     }
158     // 2.如果pos不为0,让h向后走,然后pos--, (退出条件是链表到结尾)
159     while (h->next) {
160         if (pos != 0) {
161             h = h->next;
162             pos--;
163         } else {
164             // 3.让tmp指向h->next节点
165             tmp = h->next;
166             // 4.让h->next指向tmp->next
167             h->next = tmp->next;
168             // 5.释放节点内存
169             data = tmp->data;
170             if (tmp != NULL) {
171                 free(tmp);
172                 tmp = NULL;
173             }
174             return data;
175         }
176     }
177
178     printf("%s pos right error\n", __func__);
179     return (datatype)-1;
180 }
181
182 datatype LinkListCheckDataByPos(linklist_t* h, int pos)
183 {
184     // 1.判断位置（左侧）合法性
185     if (pos < 0) {
186         printf("%s pos left error\n", __func__);
187         return (datatype)-1;
188     }
189     // 2.如果pos不为0,让h向后走,然后pos--, (退出条件是链表到结尾)
190     while (h->next) {
191         if (pos != 0) {
192             h = h->next;
193             pos--;
194         } else {
```

```

195         return h->next->data;
196     }
197 }
198
199 printf("%s pos right error\n", __func__);
200 return (datatype)-1;
201 }
202 int LinkedListUpdateDataByPos(linklist_t* h, int pos, datatype data)
203 {
204     // 1.判断位置（左侧）合法性
205     if (pos < 0) {
206         printf("%s pos left error\n", __func__);
207         return -1;
208     }
209     // 2.如果pos不为0, 让h向后走, 然后pos--, (退出条件是链表到结尾)
210     while (h->next) {
211         if (pos != 0) {
212             h = h->next;
213             pos--;
214         } else {
215             h->next->data = data;
216             return 0;
217         }
218     }
219
220     printf("%s pos right error\n", __func__);
221     return -1;
222 }
223
224 void LinkedListReverse(linklist_t* h)
225 {
226     linklist_t *tmp, *s;
227     tmp = h->next;
228     h->next = NULL;
229     while (tmp) {
230         s = tmp;
231         tmp = tmp->next;
232         s->next = h->next;
233         h->next = s;
234     }
235 }
236 void LinkedListInsertSort(linklist_t* h)
237 {
238     linklist_t *tmp, *s, *th = h;
239     tmp = h->next;
240     h->next = NULL;
241     while (tmp) {
242         s = tmp;
243         tmp = tmp->next;
244
245         while (h->next != NULL && h->next->data < s->data) {
246             h = h->next;
247         }
248
249         s->next = h->next;
250         h->next = s;
251         h = th;
252     }
253 }

```

main.c

```

1  #include "linklist.h"
2
3  int main(int argc, const char* argv[])
4  {
5      linklist_t* h;
6      h = LinkedListCreate();
7      if (h == NULL) {
8          return -1;
9      }
10
11     // LinkedListInsertHead(h, 111);
12     // LinkedListInsertHead(h, 222);
13     // LinkedListInsertHead(h, 333);
14     // LinkedListInsertHead(h, 444);
15     // LinkedListShow(h);
16
17     // LinkedListInsertTail(h, 111);
18     // LinkedListInsertTail(h, 222);
19     // LinkedListInsertTail(h, 333);
20     // LinkedListShow(h);
21     // LinkedListInsertByPos(h, 4, 777);

```

```
22 // LinkListShow(h);
23 // printf("delete head = %d\n", LinkListDeleteHead(h));
24 // LinkListShow(h);
25 // printf("delete head = %d\n", LinkListDeleteHead(h));
26 // LinkListShow(h);
27
28 // printf("delete tail = %d\n", LinkListDeleteTail(h));
29 // LinkListShow(h);
30 // printf("delete tail = %d\n", LinkListDeleteTail(h));
31 // LinkListShow(h);
32 // printf("delete tail = %d\n", LinkListDeleteTail(h));
33 // LinkListShow(h);
34 // printf("delete tail = %d\n", LinkListDeleteTail(h));
35 // LinkListShow(h);
36
37 // LinkListDeleteByPos(h,2);
38 // LinkListShow(h);
39 // printf("check data = %d\n",LinkListCheckDataByPos(h,3));
40 // LinkListUpdateDataByPos(h,2,555);
41 // LinkListShow(h);
42 // LinkListReverse(h);
43 // LinkListShow(h);
44
45 LinkListInsertTail(h, 10);
46 LinkListInsertTail(h, 5);
47 LinkListInsertTail(h, 20);
48 LinkListInsertTail(h, 40);
49 LinkListShow(h);
50 LinkListInsertSort(h);
51 LinkListShow(h);
52 return 0;
53 }
```

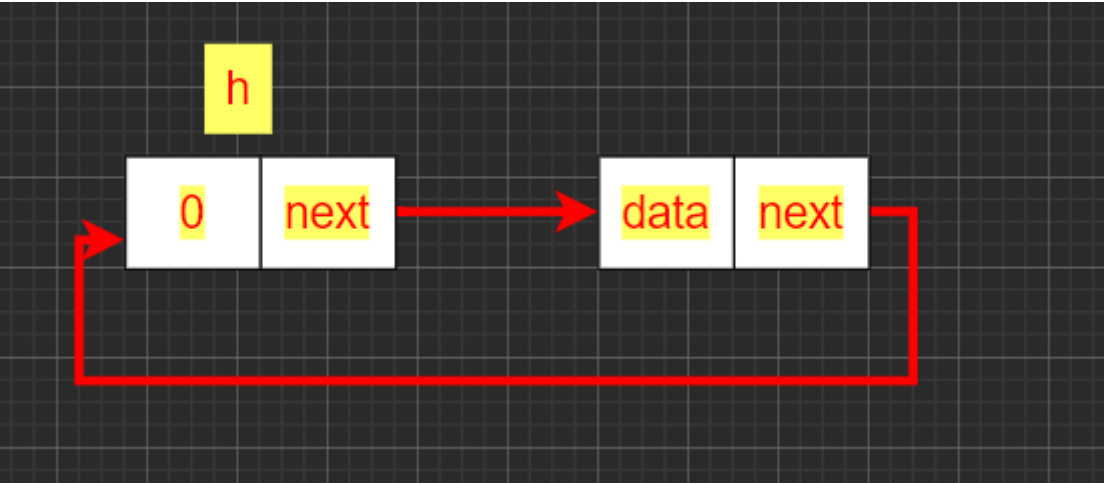
2.单向循环链表

2.1单向循环链表的特点

单链表的尾节点的指针指向的是NULL，如果有一个指针指向了尾结点那它就不能重新指向前面的节点了。为了改善这种问题将尾节点的next指针指向头节点，此时的单链表就变成了单向循环链表。

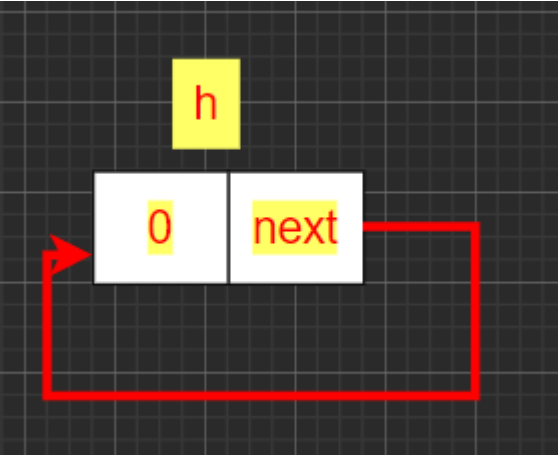
2.2单向循环链表结构

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



2.3单向循环链表的常见操作

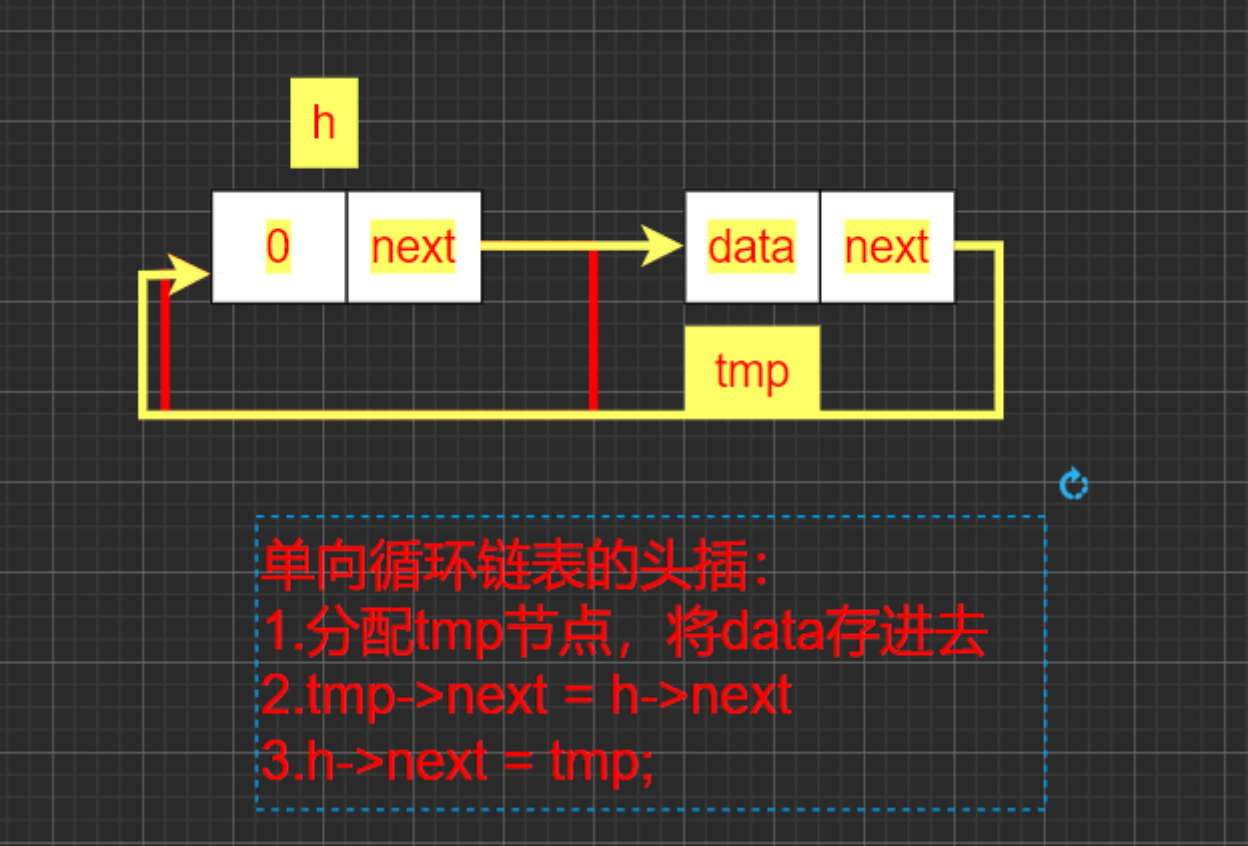
2.3.1创建单向循环链表



```
1 looplist_t* LoopListCreate(void)
2 {
3     looplist_t* h;
```

```
4
5     h = (looplist_t*)malloc(sizeof(*h));
6     if (h == NULL) {
7         printf("%s no memory\n", __func__);
8         return NULL;
9     }
10    h->data = (datatype)0;
11    h->next = h;
12
13    return h;
14 }
```

2.3.2单向循环链表的头插



```
1
2 int LoopListInsertHead(looplist_t* h, datatype data)
3 {
4     looplist_t* tmp;
5
6     tmp = (looplist_t*)malloc(sizeof(*tmp));
7     if (tmp == NULL) {
8         printf("%s no memory\n", __func__);
9         return -1;
10    }
11    tmp->data = data;
12
13    tmp->next = h->next;
14    h->next = tmp;
15
16    return 0;
17 }
```

2.3.3单向循环链表的遍历

```
1 void LoopListShow(looplist_t *h)
2 {
3     looplist_t *th = h;
4     while(h->next != th){
5         printf("-%d",h->next->data);
6         h = h->next;
7     }
8     printf("-\n");
9 }
```

单向循环链表的尾插

单向循环链表的位置插入

2.3.4单向循环链表判空

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

2.3.5单向循环链表头删

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

单向循环链表尾删

单向循环链表位置删

2.3.6单向循环链根据位置查询数据

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

2.3.7单向循环链根据位置更新数据

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

2.4单向循环链表整体代码

looplist.h

```
1 #ifndef __LINKLOOP_H__
2 #define __LINKLOOP_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* next;
11  } looplist_t;
12
13  looplist_t* LoopListCreate(void);
14  int LoopListInsertHead(looplist_t* h, datatype data);
15  void LoopListShow(looplist_t *h);
16  int LoopListIsEmpty(looplist_t *h);
17  datatype LoopListDeleteHead(looplist_t *h);
18  datatype LoopListCheckDataByPos(looplist_t*h,int pos);
19  int LoopListUpdateDataByPos(looplist_t*h,int pos,datatype data);
20  #endif

```

looplist.c

```

1  #include "looplist.h"
2
3  looplist_t* LoopListCreate(void)
4  {
5      looplist_t* h;
6
7      h = (looplist_t*)malloc(sizeof(*h));
8      if (h == NULL) {
9          printf("%s no memory\n", __func__);
10         return NULL;
11     }
12     h->data = (datatype)0;
13     h->next = h;
14
15     return h;
16 }
17 int LoopListInsertHead(looplist_t* h, datatype data)
18 {
19     looplist_t* tmp;
20
21     tmp = (looplist_t*)malloc(sizeof(*tmp));
22     if (tmp == NULL) {
23         printf("%s no memory\n", __func__);
24         return -1;
25     }
26     tmp->data = data;
27
28     tmp->next = h->next;
29     h->next = tmp;
30
31     return 0;
32 }
33 void LoopListShow(looplist_t* h)
34 {
35     looplist_t* th = h;
36     while (h->next != th) {
37         printf("%d", h->next->data);
38         h = h->next;
39     }
40     printf("-\n");
41 }
42 int LoopListIsEmpty(looplist_t* h)
43 {
44     return (h->next == h) ? 1 : 0;
45 }
46 datatype LoopListDeleteHead(looplist_t* h)
47 {
48     datatype data;
49     looplist_t* tmp;
50     if (LoopListIsEmpty(h)) {
51         printf("%s is empty\n", __func__);
52         return (datatype)-1;
53     }
54
55     tmp = h->next;
56     h->next = tmp->next;
57
58     data = tmp->data;
59     if (tmp != NULL) {
60         free(tmp);
61         tmp = NULL;
62     }
63
64     return data;
65 }
66
67 datatype LoopListCheckDataByPos(looplist_t* h, int pos)

```



```

68 {
69     looplist_t* th = h;
70     if (pos < 0) {
71         printf("%s pos left error\n", __func__);
72         return (datatype)-1;
73     }
74     while (h->next != th) {
75         if (pos != 0) {
76             h = h->next;
77             pos--;
78         } else {
79             return h->next->data;
80         }
81     }
82     printf("%s pos right error\n", __func__);
83     return (datatype)-1;
84 }
85 int LoopListUpdateDataByPos(looplist_t* h, int pos, datatype data)
86 {
87     looplist_t* th = h;
88     if (pos < 0) {
89         printf("%s pos left error\n", __func__);
90         return -1;
91     }
92     while (h->next != th) {
93         if (pos != 0) {
94             h = h->next;
95             pos--;
96         } else {
97             h->next->data = data;
98             return 0;
99         }
100     }
101     printf("%s pos right error\n", __func__);
102     return -1;
103 }

```

main.c

```

1  #include "looplist.h"
2
3  int main(int argc, const char* argv[])
4  {
5      looplist_t* h;
6
7      h = LoopListCreate();
8      if (h == NULL) {
9          return -1;
10     }
11     LoopListInsertHead(h,1);
12     LoopListInsertHead(h,2);
13     LoopListInsertHead(h,3);
14     LoopListInsertHead(h,4);
15     LoopListShow(h);
16     // LoopListDeleteHead(h);
17     // LoopListDeleteHead(h);
18     // LoopListShow(h);
19     printf("check data = %d\n",LoopListCheckDataByPos(h,2));
20     LoopListUpdateDataByPos(h,2,666);
21     printf("check data = %d\n",LoopListCheckDataByPos(h,2));
22     LoopListShow(h);
23     return 0;
24 }

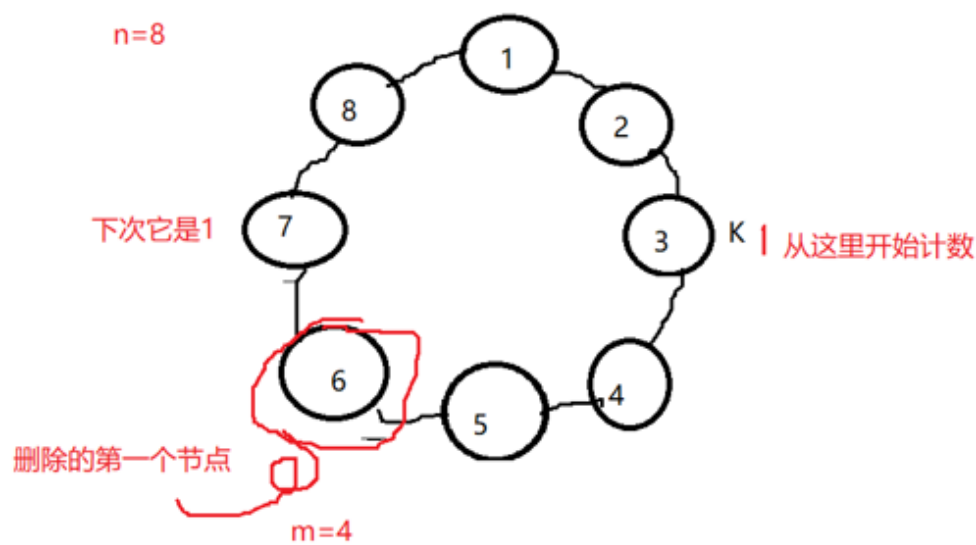
```

3.作业

joseph约瑟夫问题

设编号分别为：1, 2, ..., n的n个人围坐一圈。约定序号为k
($1 \leq k \leq n$) 的人从1开始计数，数到m的那个人出列，他的下一位
又从1开始计数，数到m的那个人又出列，依次类推，直到所有人出
列为止。例如：设n=8, k=3, m=4时
最终得到的出列序列为：(6, 2, 7, 4, 3, 5, 1, 8)

提示：使用单向循环链表实现



最终得到的出列序列为：(6, 2, 7, 4, 3, 5, 1, 8)

```
1 looplist_t* LoopListCutHead(looplist_t* h)
2 {
3     looplist_t* th = h;
4
5     if (LoopListIsEmpty(h)) {
6         printf("%s is empty\n", __func__);
7         return NULL;
8     }
9
10    while (h->next != th) {
11        h = h->next;
12    }
13
14    h->next = th->next;
15    if (th != NULL) {
16        free(th);
17        th = NULL;
18    }
19
20    return h->next;
21 }
22
23 void LoopListNoHeadShow(looplist_t* h)
24 {
25     looplist_t* th = h;
26     while (h->next != th) {
27         printf("%d", h->data);
28         h = h->next;
29     }
30     printf("%d-\n", h->data);
31 }
32 // n:单向循环链表中成员个数
33 // k:从那个人开始数
34 // m:数到m的人出列
35 void joseph(int n, int k, int m)
36 {
37     int pos = k + m - 3;
38     datatype data;
39     looplist_t* h;
40     // 1.创建单向循环链表
41     h = LoopListCreate();
42     if (h == NULL) {
43         printf("%s create loop list error\n", __func__);
44         return;
45     }
```

```
46 // 2.向单向循环链表插入数据
47 for (int i = 0; i < n; i++) {
48     LoopListInsertHead(h, n - i);
49 }
50 // 3.遍历单向循环链表
51 LoopListShow(h);
52
53 // 4.删除单向循环链表的头
54 h = LoopListCutHead(h);
55
56 // 5.无头单链表的遍历
57 LoopListNoHeadShow(h);
58
59 // 6.按照规则进行成员删除
60 if (pos == -1) //如果k和m都传递1，pos就是-1，如果是-1就站在尾结点删除第一个节点
61     pos = n - 1;
62 while (1) {
63     if (pos != 0) {
64         h = h->next;
65         pos--;
66     } else {
67         //获取h下一个节点的数据，如果是-1退出循环
68         if ((data = LoopListCheckDataByPos(h, 0)) == -1)
69             break;
70         //将获取到的数据打印到终端
71         printf(" %d", data);
72         //删除h的下一个节点
73         LoopListDeleteHead(h);
74         //对pos重新赋值
75         pos = m - 1;
76     }
77 }
78
79 printf(" %d\n", h->data); //访问最后一个节点的数据
80 if (h != NULL) { //释放最后一个节点
81     free(h);
82     h = NULL;
83 }
84 }
```