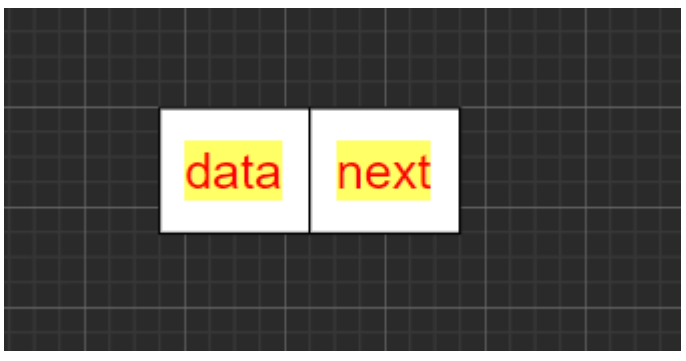
1.单链表

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



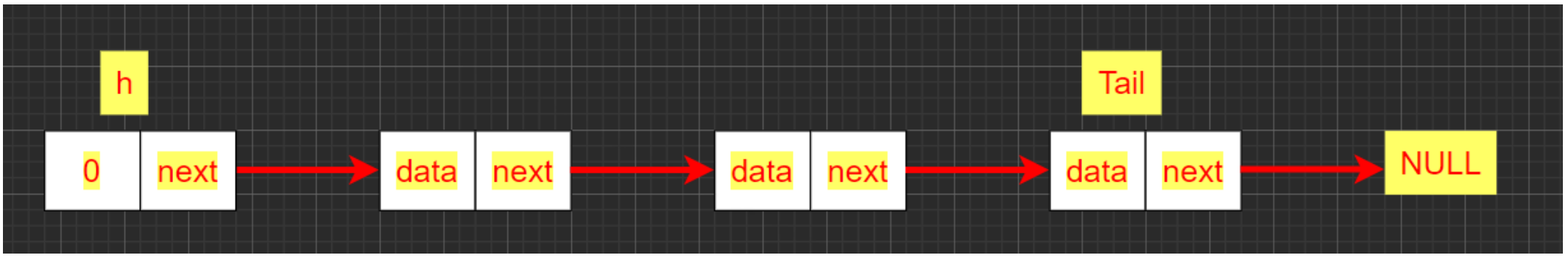
data:节点的数据域

next:节点的指针域

2.单链表节点结构体封装

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

3.链表组成



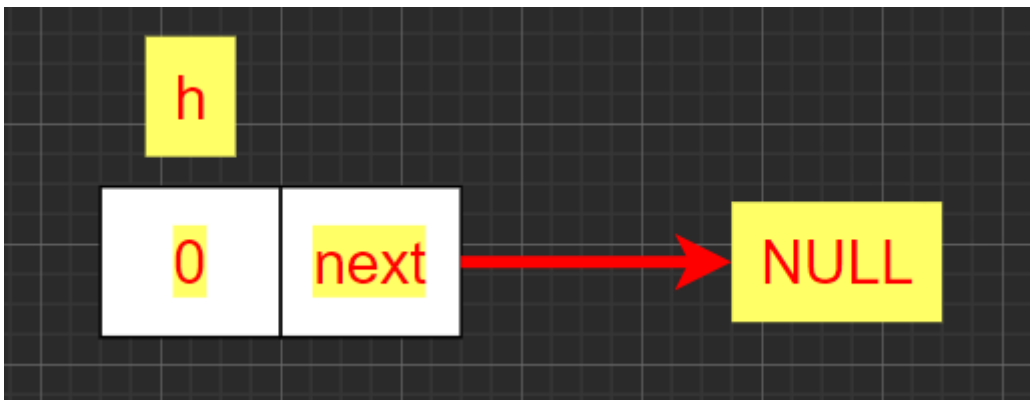
h：单链表的头节点，只有后继节点，没有前驱节点，h->data存放的是0(不存数据)

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。

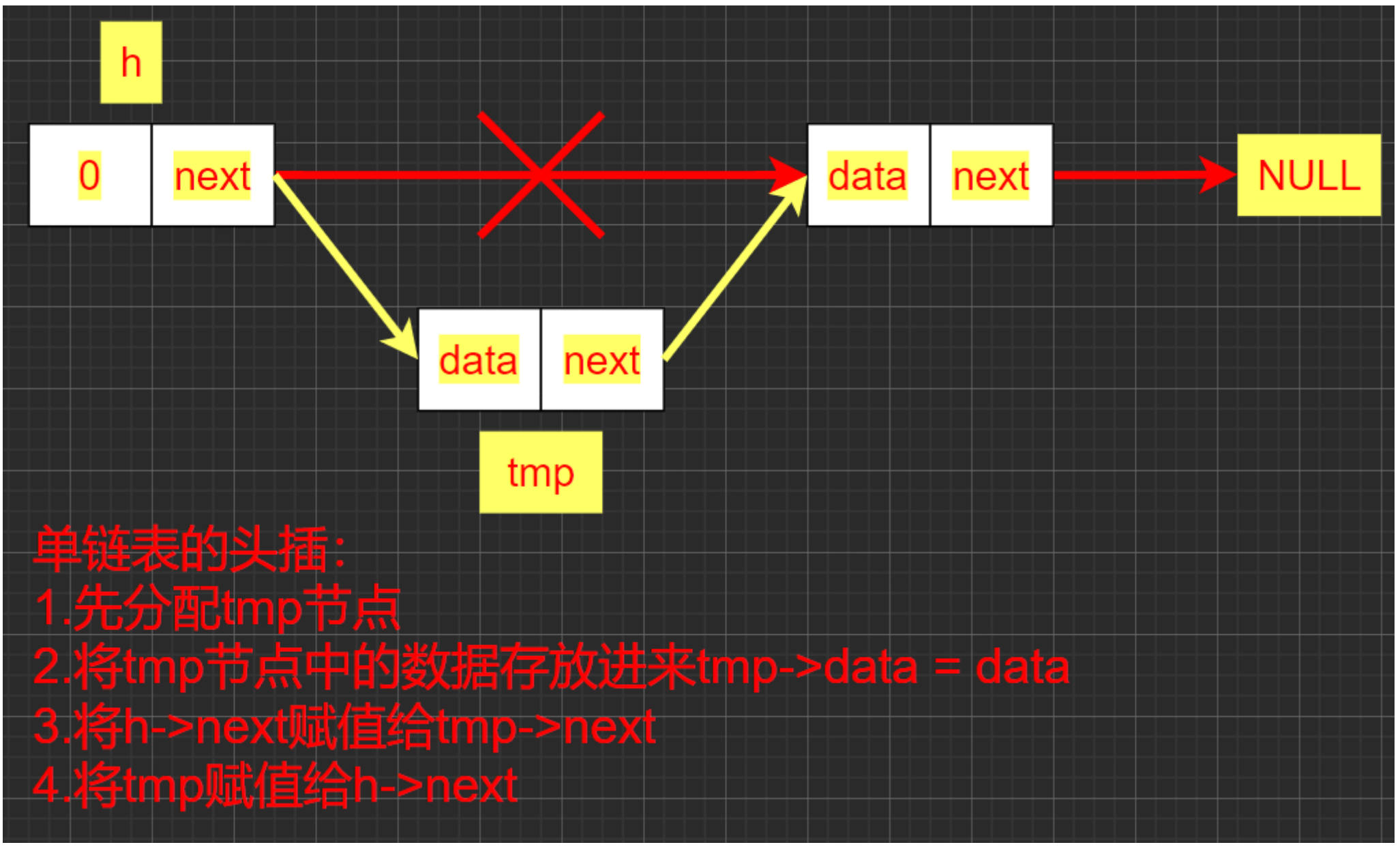
1.3单链表的常见操作

1.3.1创建单链表




```c
linklist_t* LinkListCreate(void)
{
    linklist_t* h = NULL;
    // 1.分配节点的内存
    h = (linklist_t*)malloc(sizeof(*h));
    if (h == NULL) {
        printf("%s malloc memory error\n", __func__);
        return NULL;
    }
    // 2.对节点中的成员初始化
    h->data = (datatype)0;
    h->next = NULL;

    // 3.将节点的地址返回
    return h;
}
```

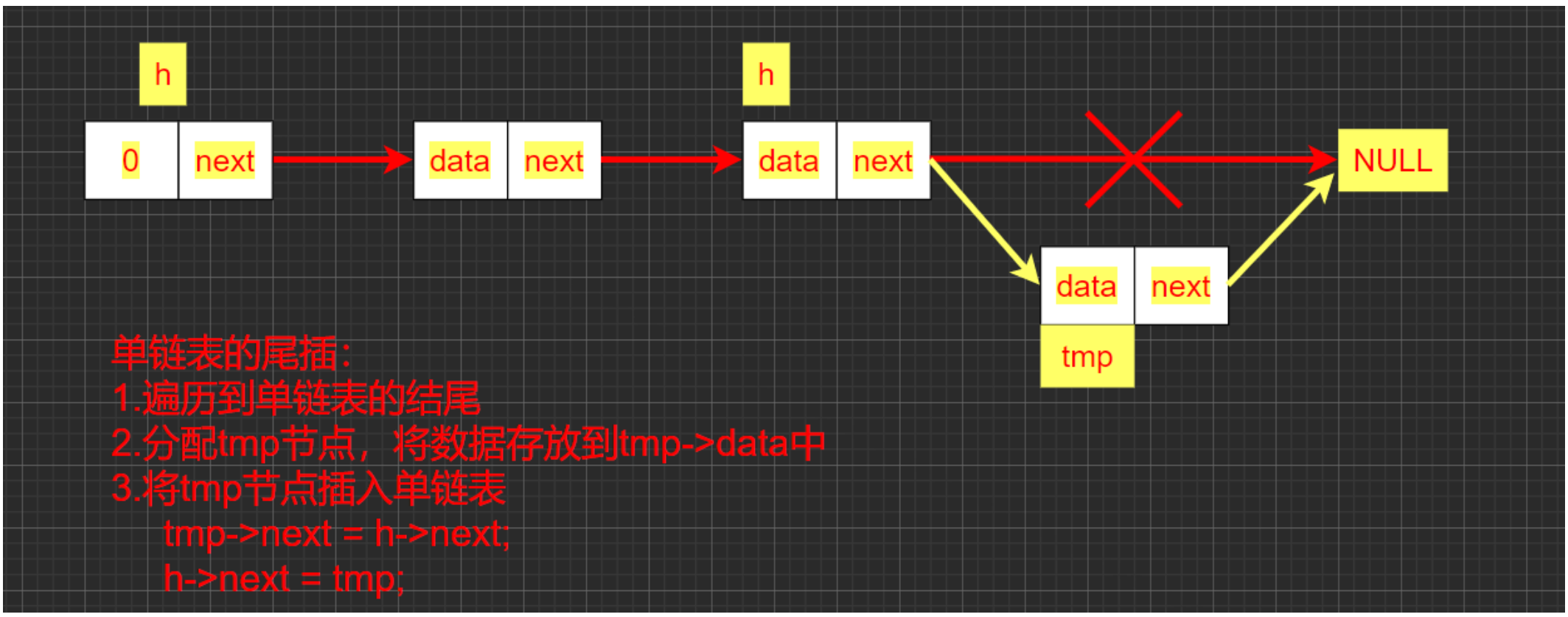
1.3.2单链表的头插




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

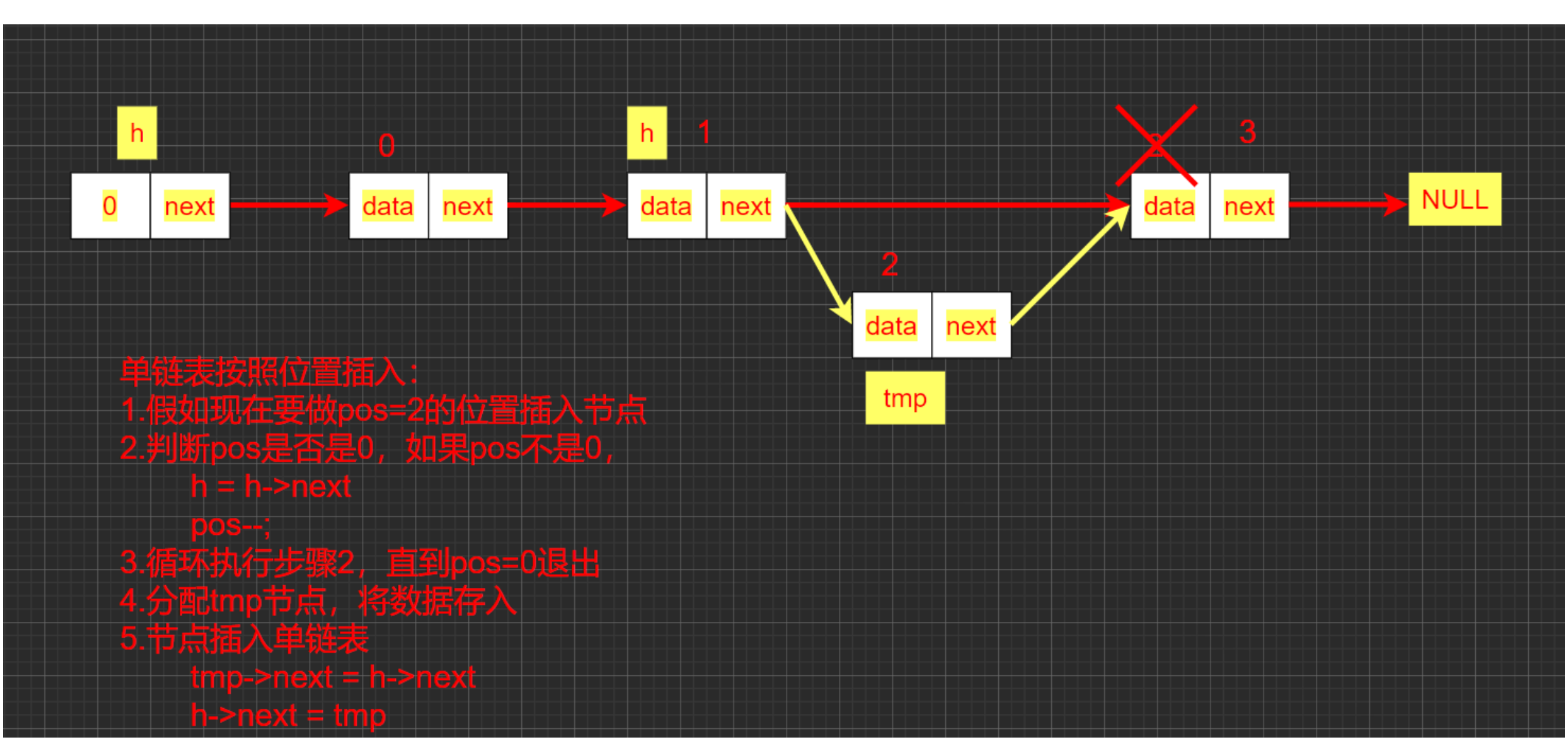
```
    tmp->data = data;
    // 3.将tmp节点头插到链表中
    tmp->next = h->next;
    h->next = tmp;

    return 0;
}
```

1.3.3单链表的尾插




```
int LinkListInsertTail(linklist_t* h, datatype data)
{
    linklist_t* tmp;
    // 1.遍历到链表的结尾
    while (h->next)
        h = h->next;
    // 2.分配tmp节点内存，将data存放进来
    tmp = (linklist_t*)malloc(sizeof(*tmp));
    if (tmp == NULL) {
        printf("%s malloc memory error\n", __func__);
        return -1;
    }
    tmp->data = data;
    // 3.将tmp节点插入单链表
    tmp->next = h->next;
    h->next = tmp;

    return 0;
}
```

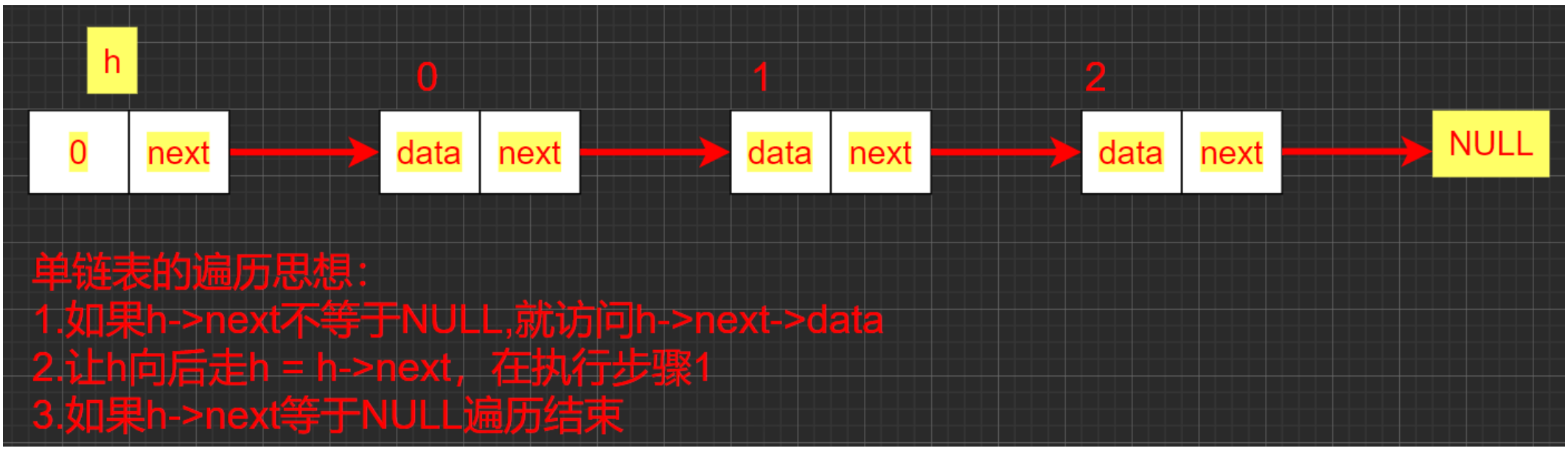
1.3.4单链表任意位置插入节点




```
int LinkListInsertByPos(linklist_t* h, int pos, datatype data)
```

```
{
    linklist_t* tmp;
    // 1.判断位置（左侧）合法性
    if (pos < 0) {
        printf("%s pos left error\n", __func__);
        return -1;
    }
    // 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)
    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
        if (pos != 0) {
            h = h->next;
            pos--;
        } else {
            // 3.分配tmp节点，将数据存入
            tmp = (linklist_t*)malloc(sizeof(*tmp));
            if (tmp == NULL) {
                printf("%s malloc memory error\n", __func__);
                return -1;
            }
            tmp->data = data;
            // 4.将tmp节点插入单链表
            tmp->next = h->next;
            h->next = tmp;
            return 0;
        }
    }

    printf("%s pos right error\n", __func__);
    return -1;
}
```

1.3.5单链表的遍历



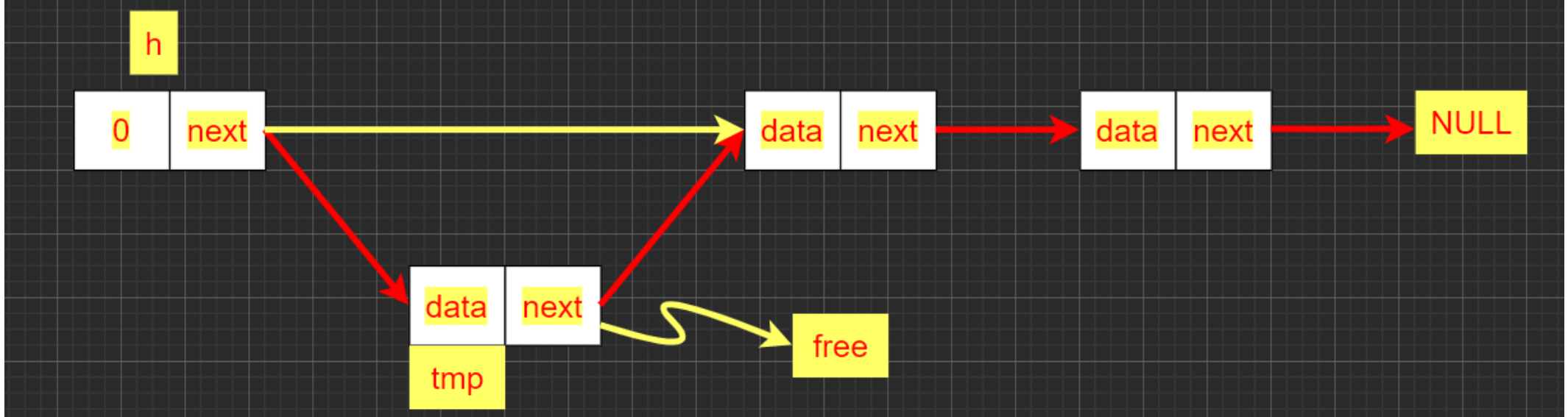

```
void LinkListShow(linklist_t* h)
{
    while(h->next){
        printf("-%d",h->next->data);
        h = h->next;
    }
    printf("-\n");
}
```

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

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

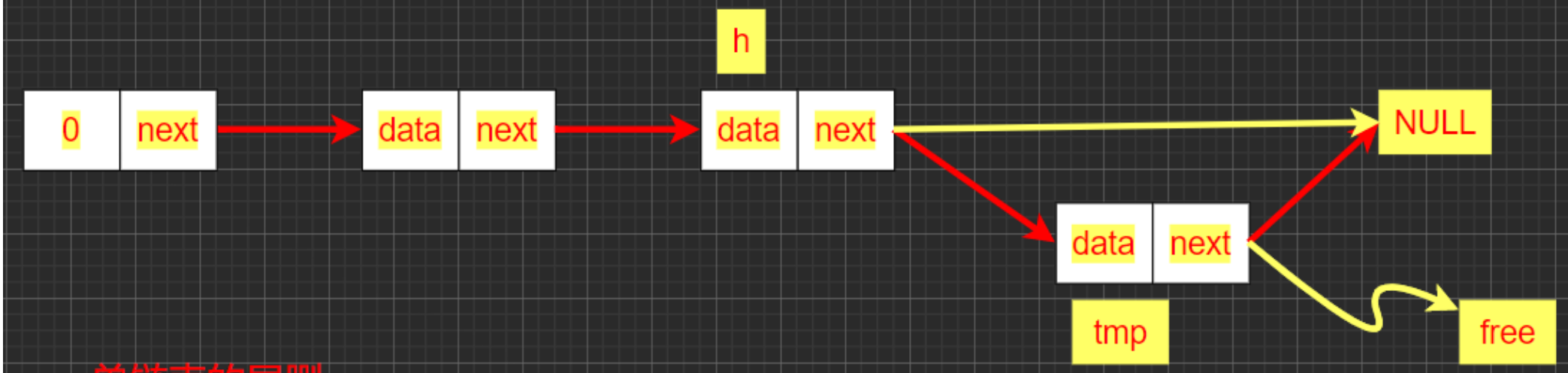
1.3.7单链表头删



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;

```c
datatype LinkListDeleteHead(linklist_t* h)
{
    datatype data;
    linklist_t* tmp;
    // 1.判空
    if (LinkListIsEmpty(h)) {
        printf("%s is empty\n", __func__);
        return (datatype)-1;
    }
    // 2.让tmp指向h->next节点
    tmp = h->next;
    // 3.让h->next指向tmp->next
    h->next = tmp->next;
    // 4.释放节点内存
    data = tmp->data;
    if (tmp != NULL) {
        free(tmp);
        tmp = NULL;
    }
    return data;
}
```

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
```

```c
datatype LinkListDeleteTail(linklist_t* h)
{
    datatype data;
    linklist_t* tmp;
    // 1.判空
    if (LinkListIsEmpty(h)) {
        printf("%s is empty\n", __func__);
        return (datatype)-1;
    }
    // 2.让h走到倒数第二个节点停下
    while(h->next->next)
        h = h->next;
    // 3.让tmp指向h->next节点
    tmp = h->next;
    // 4.让h->next指向tmp->next
    h->next = tmp->next;
    // 5.释放节点内存
    data = tmp->data;
    if (tmp != NULL) {
        free(tmp);
        tmp = NULL;
    }
    return data;
}
```

> 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

```c
datatype LinkListDeleteByPos(linklist_t* h, int pos)
{
    linklist_t* tmp;
    datatype data;
    // 1.判断位置（左侧）合法性
    if (pos < 0) {
        printf("%s pos left error\n", __func__);
        return -1;
    }
    // 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)
    while (h->next) {
        if (pos != 0) {
            h = h->next;
            pos--;
        } else {
            // 3.让tmp指向h->next节点
            tmp = h->next;
            // 4.让h->next指向tmp->next
            h->next = tmp->next;
            // 5.释放节点内存
```

```
            data = tmp->data;
            if (tmp != NULL) {
                free(tmp);
                tmp = NULL;
            }
            return data;
        }
    }

    printf("%s pos right error\n", __func__);
    return -1;
}
```

1.3.10单链表根据位置查询数据

```
datatype LinkListCheckDataByPos(linklist_t* h, int pos)
{
    // 1.判断位置（左侧）合法性
    if (pos < 0) {
        printf("%s pos left error\n", __func__);
        return (datatype)-1;
    }
    // 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)
    while (h->next) {
        if (pos != 0) {
            h = h->next;
            pos--;
        } else {
            return h->next->data;
        }
    }

    printf("%s pos right error\n", __func__);
    return (datatype)-1;
}
```

1.3.11单链表根据位置更新数据

```
int LinkListUpdateDataByPos(linklist_t* h, int pos, datatype data)
{
    // 1.判断位置（左侧）合法性
    if (pos < 0) {
        printf("%s pos left error\n", __func__);
        return -1;
    }
    // 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)
    while (h->next) {
        if (pos != 0) {
            h = h->next;
            pos--;
        } else {
            h->next->data = data;
            return 0;
        }
    }

    printf("%s pos right error\n", __func__);
    return -1;
}
```

1.3.12单链表逆序

h-10-30-20-40

h-40-20-30-10

```
void LinkListReverse(linklist_t *h)
{
    linklist_t *tmp,*s;
    tmp = h->next;
    h->next = NULL;
    while(tmp){
        s = tmp;
        tmp = tmp->next;
        s->next = h->next;
        h->next = s;
    }
}
```

1.3.13单链表排序(直接插入排序)

tmp-5-30-20-40

h-10-NULL

```c
void LinkListInsertSort(linklist_t* h)
{
    linklist_t *tmp, *s, *th = h;
    tmp = h->next;
    h->next = NULL;
    while (tmp) {
        s = tmp;
        tmp = tmp->next;
        while (h->next != NULL && h->next->data < s->data) {
            h = h->next;
        }
        s->next = h->next;
        h->next = s;
        h = th;
    }
}
```

1.4单链表整体代码

linklist.h

```c
#ifndef __LINKLIST_H__
#define __LINKLIST_H__

#include <stdio.h>
#include <stdlib.h>

#define datatype int
typedef struct node {
    datatype data;
    struct node* next;
} linklist_t;

linklist_t* LinkListCreate(void);
int LinkListInsertHead(linklist_t* h, datatype data);
int LinkListInsertTail(linklist_t* h, datatype data);
int LinkListInsertByPos(linklist_t* h, int pos, datatype data);
void LinkListShow(linklist_t* h);
int LinkListIsEmpty(linklist_t *h);
datatype LinkListDeleteHead(linklist_t *h);
datatype LinkListDeleteTail(linklist_t *h);
datatype LinkListDeleteByPos(linklist_t *h,int pos);
datatype LinkListCheckDataByPos(linklist_t *h,int pos);
int LinkListUpdateDataByPos(linklist_t *h,int pos,datatype data);
void LinkListReverse(linklist_t *h);
void LinkListInsertSort(linklist_t *h);
#endif
```

linklist.c

```c
#include "linklist.h"

linklist_t* LinkListCreate(void)
{
    linklist_t* h;
    // 1.分配节点的内存
    h = (linklist_t*)malloc(sizeof(*h));
    if (h == NULL) {
        printf("%s malloc memory error\n", __func__);
        return NULL;
    }
    // 2.对节点中的成员初始化
    h->data = (datatype)0;
    h->next = NULL;

    // 3.将节点的地址返回
    return h;
}

int LinkListInsertHead(linklist_t* h, datatype data)
{
    linklist_t* tmp;
    // 1.分配tmp节点
    tmp = (linklist_t*)malloc(sizeof(*tmp));
    if (tmp == NULL) {
        printf("%s malloc memory error\n", __func__);
```

```
        return -1;
    }
    // 2.将数据存入
    tmp->data = data;
    // 3.将tmp节点头插到链表中
    tmp->next = h->next;
    h->next = tmp;

    return 0;
}
int LinkListInsertTail(linklist_t* h, datatype data)
{
    linklist_t* tmp;
    // 1.遍历到链表的结尾
    while (h->next)
        h = h->next;
    // 2.分配tmp节点内存，将data存放进来
    tmp = (linklist_t*)malloc(sizeof(*tmp));
    if (tmp == NULL) {
        printf("%s malloc memory error\n", __func__);
        return -1;
    }
    tmp->data = data;
    // 3.将tmp节点插入单链表
    tmp->next = h->next;
    h->next = tmp;

    return 0;
}

int LinkListInsertByPos(linklist_t* h, int pos, datatype data)
{
    linklist_t* tmp;
    // 1.判断位置（左侧）合法性
    if (pos < 0) {
        printf("%s pos left error\n", __func__);
        return -1;
    }
    // 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)
    while (h) {
        if (pos != 0) {
            h = h->next;
            pos--;
        } else {
            // 3.分配tmp节点，将数据存入
            tmp = (linklist_t*)malloc(sizeof(*tmp));
            if (tmp == NULL) {
                printf("%s malloc memory error\n", __func__);
                return -1;
            }
            tmp->data = data;
            // 4.将tmp节点插入单链表
            tmp->next = h->next;
            h->next = tmp;
            return 0;
        }
    }

    printf("%s pos right error\n", __func__);
    return -1;
}

void LinkListShow(linklist_t* h)
{
    while (h->next) {
        printf("-%d", h->next->data);
        h = h->next;
    }
    printf("-\n");
}
int LinkListIsEmpty(linklist_t* h)
{
    return (h->next == NULL) ? 1 : 0;
}
datatype LinkListDeleteHead(linklist_t* h)
{
    datatype data;
    linklist_t* tmp;
    // 1.判空
    if (LinkListIsEmpty(h)) {
        printf("%s is empty\n", __func__);
        return (datatype)-1;
    }
    // 2.让tmp指向h->next节点
```

```
    tmp = h->next;
    // 3.让h->next指向tmp->next
    h->next = tmp->next;
    // 4.释放节点内存
    data = tmp->data;
    if (tmp != NULL) {
        free(tmp);
        tmp = NULL;
    }

    return data;
}

datatype LinkListDeleteTail(linklist_t* h)
{
    datatype data;
    linklist_t* tmp;
    // 1.判空
    if (LinkListIsEmpty(h)) {
        printf("%s is empty\n", __func__);
        return (datatype)-1;
    }
    // 2.让h走到倒数第二个节点停下
    while (h->next->next)
        h = h->next;
    // 3.让tmp指向h->next节点
    tmp = h->next;
    // 4.让h->next指向tmp->next
    h->next = tmp->next;
    // 5.释放节点内存
    data = tmp->data;
    if (tmp != NULL) {
        free(tmp);
        tmp = NULL;
    }
    return data;
}

datatype LinkListDeleteByPos(linklist_t* h, int pos)
{
    linklist_t* tmp;
    datatype data;
    // 1.判断位置（左侧）合法性
    if (pos < 0) {
        printf("%s pos left error\n", __func__);
        return (datatype)-1;
    }
    // 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)
    while (h->next) {
        if (pos != 0) {
            h = h->next;
            pos--;
        } else {
            // 3.让tmp指向h->next节点
            tmp = h->next;
            // 4.让h->next指向tmp->next
            h->next = tmp->next;
            // 5.释放节点内存
            data = tmp->data;
            if (tmp != NULL) {
                free(tmp);
                tmp = NULL;
            }
            return data;
        }
    }

    printf("%s pos right error\n", __func__);
    return (datatype)-1;
}

datatype LinkListCheckDataByPos(linklist_t* h, int pos)
{
    // 1.判断位置（左侧）合法性
    if (pos < 0) {
        printf("%s pos left error\n", __func__);
        return (datatype)-1;
    }
    // 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)
    while (h->next) {
        if (pos != 0) {
            h = h->next;
            pos--;
        } else {
```

```
            return h->next->data;
        }
    }

    printf("%s pos right error\n", __func__);
    return (datatype)-1;
}
int LinkListUpdateDataByPos(linklist_t* h, int pos, datatype data)
{
    // 1.判断位置（左侧）合法性
    if (pos < 0) {
        printf("%s pos left error\n", __func__);
        return -1;
    }
    // 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)
    while (h->next) {
        if (pos != 0) {
            h = h->next;
            pos--;
        } else {
            h->next->data = data;
            return 0;
        }
    }

    printf("%s pos right error\n", __func__);
    return -1;
}

void LinkListReverse(linklist_t* h)
{
    linklist_t *tmp, *s;
    tmp = h->next;
    h->next = NULL;
    while (tmp) {
        s = tmp;
        tmp = tmp->next;
        s->next = h->next;
        h->next = s;
    }
}
void LinkListInsertSort(linklist_t* h)
{
    linklist_t *tmp, *s, *th = h;
    tmp = h->next;
    h->next = NULL;
    while (tmp) {
        s = tmp;
        tmp = tmp->next;

        while (h->next != NULL && h->next->data < s->data) {
            h = h->next;
        }

        s->next = h->next;
        h->next = s;
        h = th;
    }
}
```

main.c

```
#include "linklist.h"

int main(int argc, const char* argv[])
{
    linklist_t* h;
    h = LinkListCreate();
    if (h == NULL) {
        return -1;
    }

    // LinkListInsertHead(h, 111);
    // LinkListInsertHead(h, 222);
    // LinkListInsertHead(h, 333);
    // LinkListInsertHead(h, 444);
    // LinkListShow(h);

    // LinkListInsertTail(h, 111);
    // LinkListInsertTail(h, 222);
    // LinkListInsertTail(h, 333);
    // LinkListShow(h);
    // LinkListInsertByPos(h,4,777);
```

```
    // LinkListShow(h);
    // printf("delete head = %d\n", LinkListDeleteHead(h));
    // LinkListShow(h);
    // printf("delete head = %d\n", LinkListDeleteHead(h));
    // LinkListShow(h);

    // printf("delete tail = %d\n", LinkListDeleteTail(h));
    // LinkListShow(h);
    // printf("delete tail = %d\n", LinkListDeleteTail(h));
    // LinkListShow(h);
    // printf("delete tail = %d\n", LinkListDeleteTail(h));
    // LinkListShow(h);
    // printf("delete tail = %d\n", LinkListDeleteTail(h));
    // LinkListShow(h);

    // LinkListDeleteByPos(h,2);
    // LinkListShow(h);
    // printf("check data = %d\n",LinkListCheckDataByPos(h,3));
    // LinkListUpdateDataByPos(h,2,555);
    // LinkListShow(h);
    // LinkListReverse(h);
    // LinkListShow(h);

    LinkListInsertTail(h, 10);
    LinkListInsertTail(h, 5);
    LinkListInsertTail(h, 20);
    LinkListInsertTail(h, 40);
    LinkListShow(h);
    LinkListInsertSort(h);
    LinkListShow(h);
    return 0;
}
```

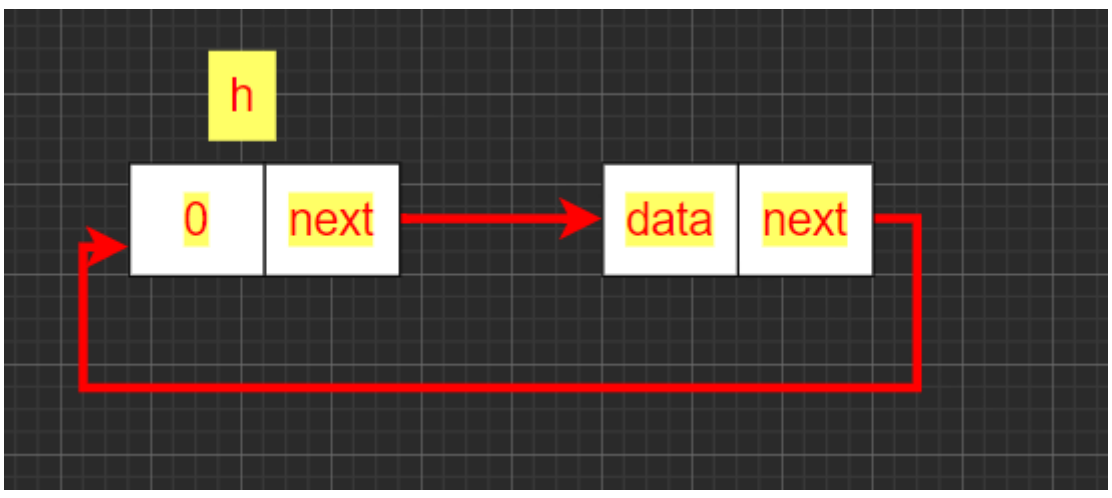
2.单向循环链表

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。

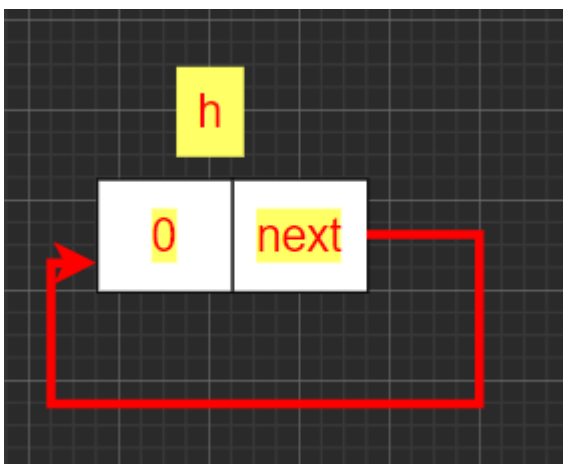
2.2单向循环链表结构

```
#define datatype int
typedef struct node{
    datatype data;
    struct node *next;
}looplist_t;
```



2.3单向循环链表的常见操作

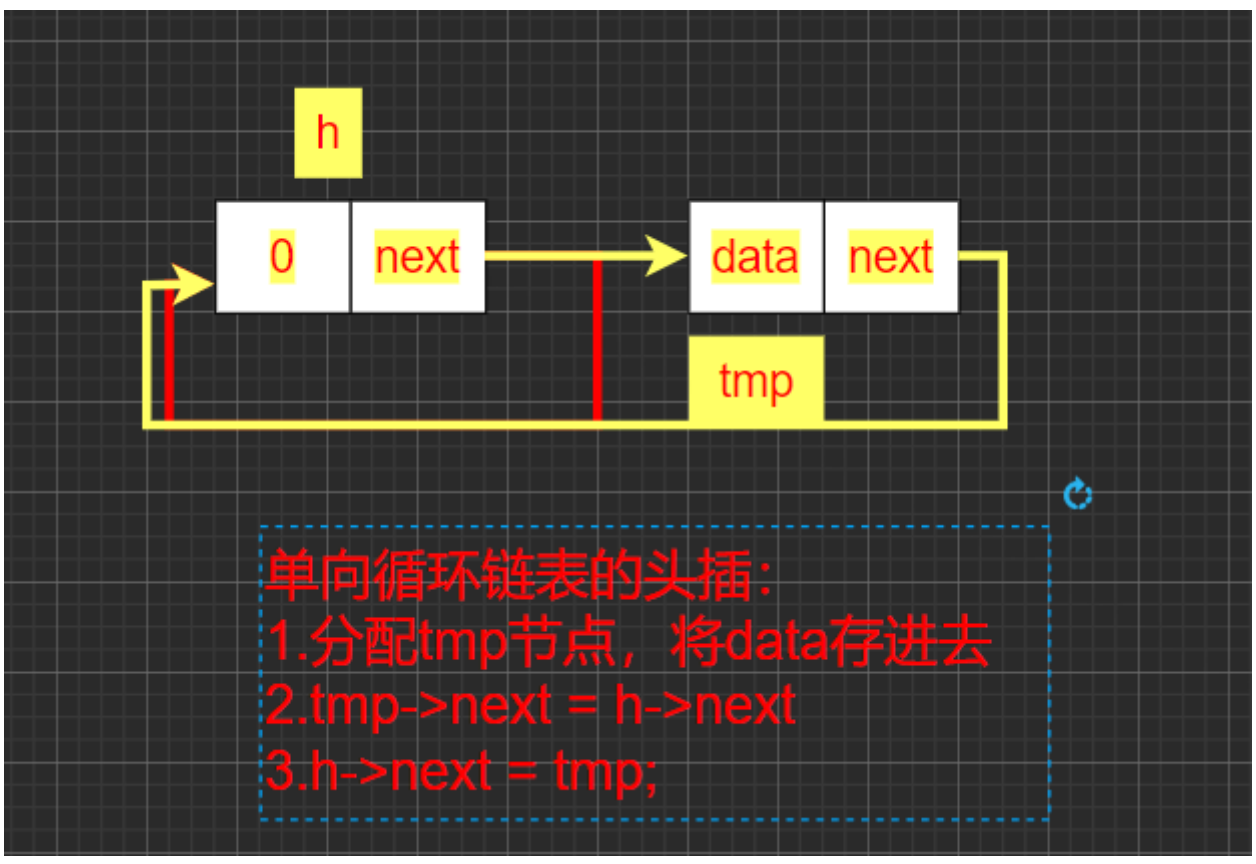
2.3.1创建单向循环链表




```
looplist_t* LoopListCreate(void)
{
    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

```c
datatype LoopListDeleteHead(looplist_t* h)
{
    datatype data;
    looplist_t* tmp;
    if (LoopListIsEmpty(h)) {
        printf("%s is empty\n", __func__);
        return (datatype)-1;
    }

    tmp = h->next;
    h->next = tmp->next;

    data = tmp->data;
    if (tmp != NULL) {
        free(tmp);
        tmp = NULL;
    }

    return data;
}
```

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

```c
datatype LoopListCheckDataByPos(looplist_t* h, int pos)
{
    looplist_t* th = h;
    if (pos < 0) {
        printf("%s pos left error\n", __func__);
        return (datatype)-1;
    }
    while (h->next != th) {
        if (pos != 0) {
            h = h->next;
            pos--;
        } else {
            return h->next->data;
        }
    }
    printf("%s pos right error\n", __func__);
    return (datatype)-1;
}
```

2.3.7单向循环链根据位置更新数据

```c
int LoopListUpdateDataByPos(looplist_t* h, int pos, datatype data)
{
    looplist_t* th = h;
    if (pos < 0) {
        printf("%s pos left error\n", __func__);
        return -1;
    }
    while (h->next != th) {
        if (pos != 0) {
            h = h->next;
            pos--;
        } else {
            h->next->data = data;
            return 0;
        }
    }
    printf("%s pos right error\n", __func__);
    return -1;
}
```

2.4单向循环链表整体代码

looplist.h

```c
#ifndef __LINKLOOP_H__
#define __LINKLOOP_H__

#include <stdio.h>
#include <stdlib.h>

#define datatype int
```

```c
typedef struct node {
    datatype data;
    struct node* next;
} looplist_t;

looplist_t* LoopListCreate(void);
int LoopListInsertHead(looplist_t* h, datatype data);
void LoopListShow(looplist_t *h);
int LoopListIsEmpty(looplist_t *h);
datatype LoopListDeleteHead(looplist_t *h);
datatype LoopListCheckDataByPos(looplist_t*h,int pos);
int LoopListUpdateDataByPos(looplist_t*h,int pos,datatype data);
#endif
```

looplist.c

```c
#include "looplist.h"

looplist_t* LoopListCreate(void)
{
    looplist_t* h;

    h = (looplist_t*)malloc(sizeof(*h));
    if (h == NULL) {
        printf("%s no memory\n", __func__);
        return NULL;
    }
    h->data = (datatype)0;
    h->next = h;

    return h;
}
int LoopListInsertHead(looplist_t* h, datatype data)
{
    looplist_t* tmp;

    tmp = (looplist_t*)malloc(sizeof(*tmp));
    if (tmp == NULL) {
        printf("%s no memory\n", __func__);
        return -1;
    }
    tmp->data = data;

    tmp->next = h->next;
    h->next = tmp;

    return 0;
}
void LoopListShow(looplist_t* h)
{
    looplist_t* th = h;
    while (h->next != th) {
        printf("-%d", h->next->data);
        h = h->next;
    }
    printf("-\n");
}
int LoopListIsEmpty(looplist_t* h)
{
    return (h->next == h) ? 1 : 0;
}
datatype LoopListDeleteHead(looplist_t* h)
{
    datatype data;
    looplist_t* tmp;
    if (LoopListIsEmpty(h)) {
        printf("%s is empty\n", __func__);
        return (datatype)-1;
    }

    tmp = h->next;
    h->next = tmp->next;

    data = tmp->data;
    if (tmp != NULL) {
        free(tmp);
        tmp = NULL;
    }

    return data;
}

datatype LoopListCheckDataByPos(looplist_t* h, int pos)
```

```
{
    looplist_t* th = h;
    if (pos < 0) {
        printf("%s pos left error\n", __func__);
        return (datatype)-1;
    }
    while (h->next != th) {
        if (pos != 0) {
            h = h->next;
            pos--;
        } else {
            return h->next->data;
        }
    }
    printf("%s pos right error\n", __func__);
    return (datatype)-1;
}
int LoopListUpdateDataByPos(looplist_t* h, int pos, datatype data)
{
    looplist_t* th = h;
    if (pos < 0) {
        printf("%s pos left error\n", __func__);
        return -1;
    }
    while (h->next != th) {
        if (pos != 0) {
            h = h->next;
            pos--;
        } else {
            h->next->data = data;
            return 0;
        }
    }
    printf("%s pos right error\n", __func__);
    return -1;
}
```

main.c

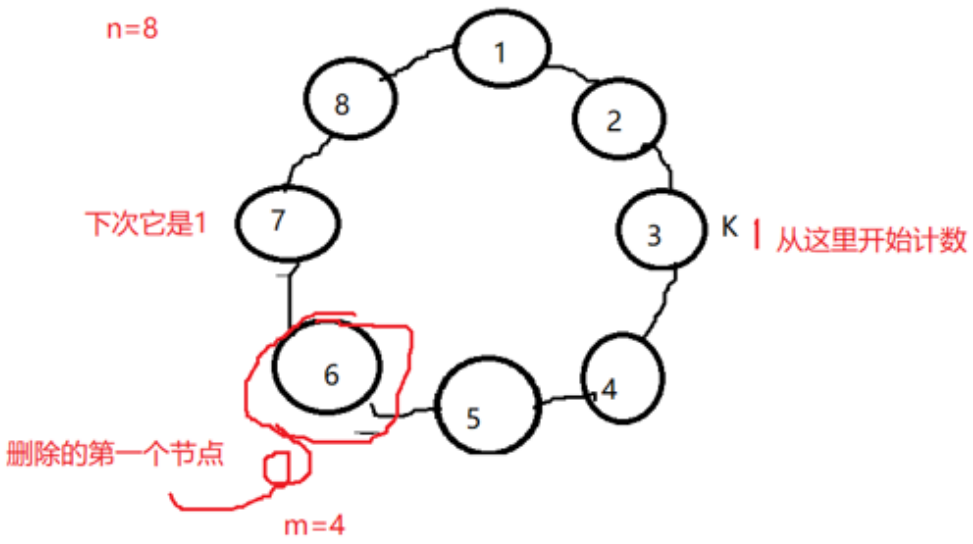
```
#include "looplist.h"

int main(int argc, const char* argv[])
{
    looplist_t* h;

    h = LoopListCreate();
    if (h == NULL) {
        return -1;
    }
    LoopListInsertHead(h,1);
    LoopListInsertHead(h,2);
    LoopListInsertHead(h,3);
    LoopListInsertHead(h,4);
    LoopListShow(h);
    // LoopListDeleteHead(h);
    // LoopListDeleteHead(h);
    // LoopListShow(h);
    printf("check data = %d\n",LoopListCheckDataByPos(h,2));
    LoopListUpdateDataByPos(h,2,666);
    printf("check data = %d\n",LoopListCheckDataByPos(h,2));
    LoopListShow(h);
    return 0;
}
```

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≤k≤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)

```c
looplist_t* LoopListCutHead(looplist_t* h)
{
    looplist_t* th = h;

    if (LoopListIsEmpty(h)) {
        printf("%s is empty\n", __func__);
        return NULL;
    }

    while (h->next != th) {
        h = h->next;
    }

    h->next = th->next;
    if (th != NULL) {
        free(th);
        th = NULL;
    }

    return h->next;
}

void LoopListNoHeadShow(looplist_t* h)
{
    looplist_t* th = h;
    while (h->next != th) {
        printf("-%d", h->data);
        h = h->next;
    }
    printf("-%d-\n", h->data);
}
// n:单向循环链表中成员个数
// k:从那个人开始数
// m:数到m的人出列
void joseph(int n, int k, int m)
{
    int pos = k + m - 3;
    datatype data;
    looplist_t* h;
    // 1.创建单向循环链表
    h = LoopListCreate();
    if (h == NULL) {
        printf("%s create loop list error\n", __func__);
        return;
    }
```

```c
    // 2.向单向循环链表插入数据
    for (int i = 0; i < n; i++) {
        LoopListInsertHead(h, n - i);
    }
    // 3.遍历单向循环链表
    LoopListShow(h);

    // 4.删除单向循环链表的头
    h = LoopListCutHead(h);

    // 5.无头单链表的遍历
    LoopListNoHeadShow(h);

    // 6.按照规则进行成员删除
    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
        pos = n - 1;
    while (1) {
        if (pos != 0) {
            h = h->next;
            pos--;
        } else {
            //获取h下一个节点的数据，如果是-1退出循环
            if ((data = LoopListCheckDataByPos(h, 0)) == -1)
                break;
            //将获取到的数据打印到终端
            printf(" %d", data);
            //删除h的下一个节点
            LoopListDeleteHead(h);
            //对pos重新赋值
            pos = m - 1;
        }
    }

    printf(" %d\n", h->data); //访问最后一个节点的数据
    if (h != NULL) {          //释放最后一个节点
        free(h);
        h = NULL;
    }
}
```