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

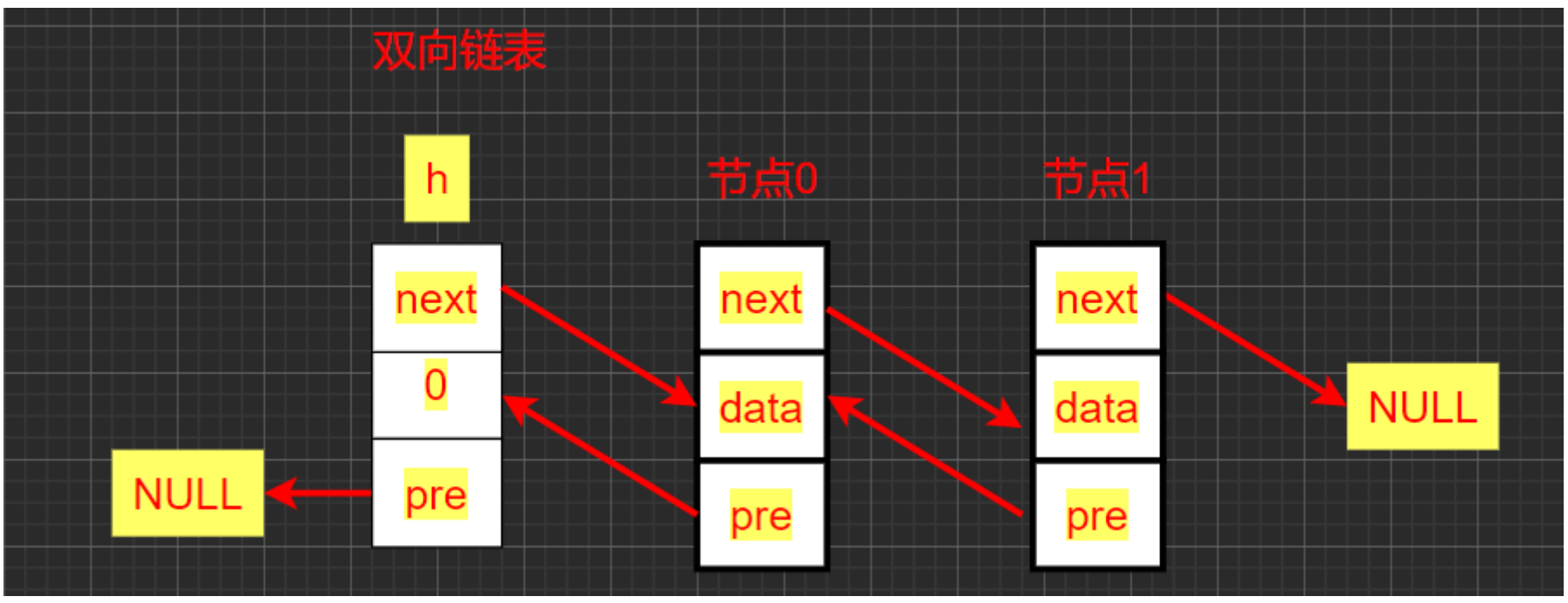
1.双向链表

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。

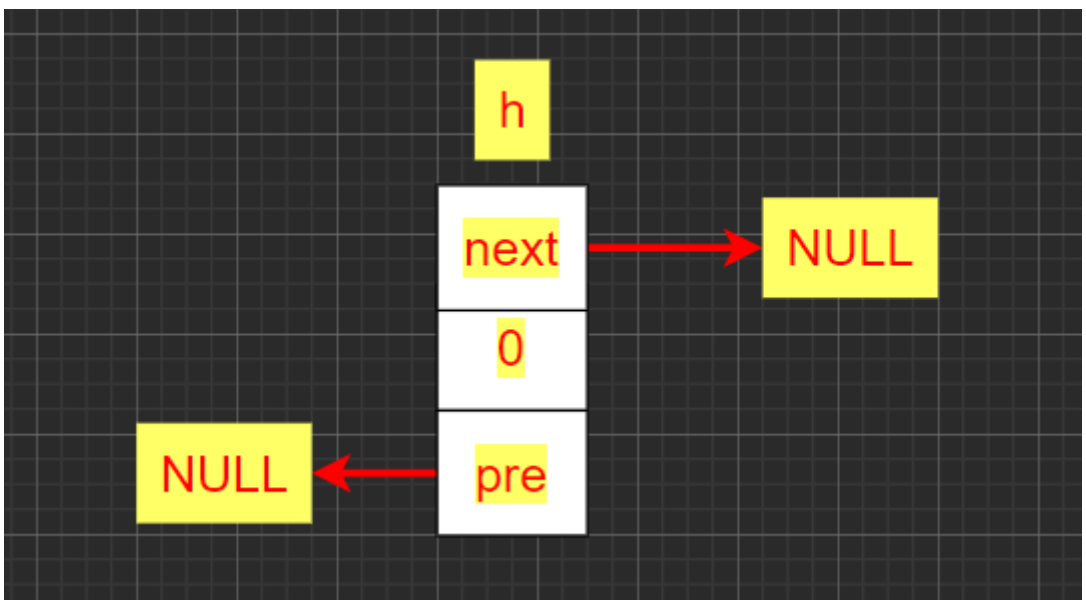
1.2双向链表的结构

```
#define datatype int
typedef struct node{
    datatype data;
    struct node *pre,*next;
}DPlist_t;
```



1.3双向链表的常见操作

1.3.1双向链表的创建



```c
DPlist_t* DPListCreate(void)
{
    DPlist_t* h;
    h = (DPlist_t*)malloc(sizeof(*h));
    if (h == NULL) {
        printf("%s malloc memory error\n", __func__);
        return NULL;
    }
    h->data = (datatype)0;
    h->next = NULL;
    h->pre = NULL;

    return h;
}
```

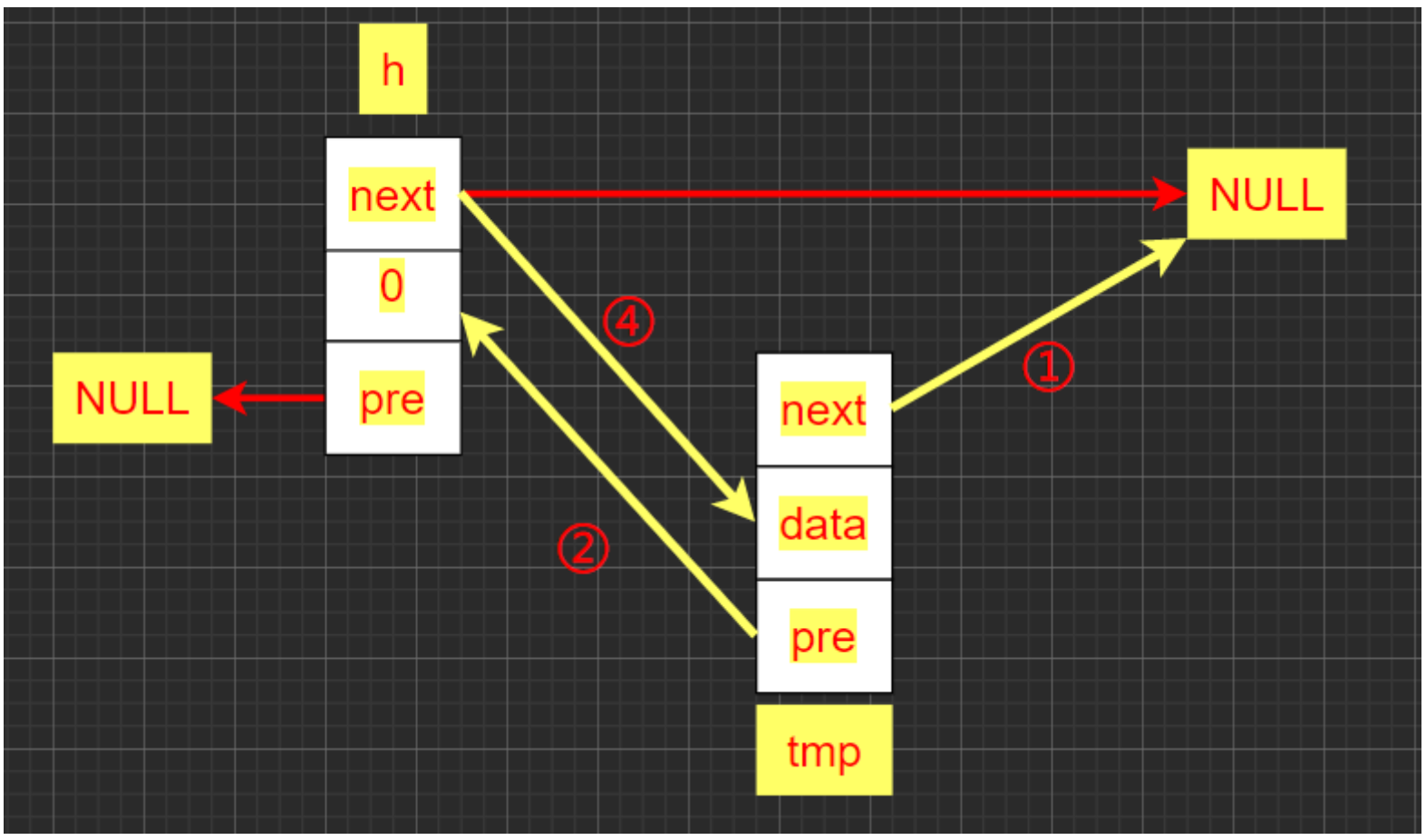
1.3.2双向链表头插

情况1：（双链表是空链表）

tmp->next = h->next;

tmp->pre = h;

h->next = tmp;



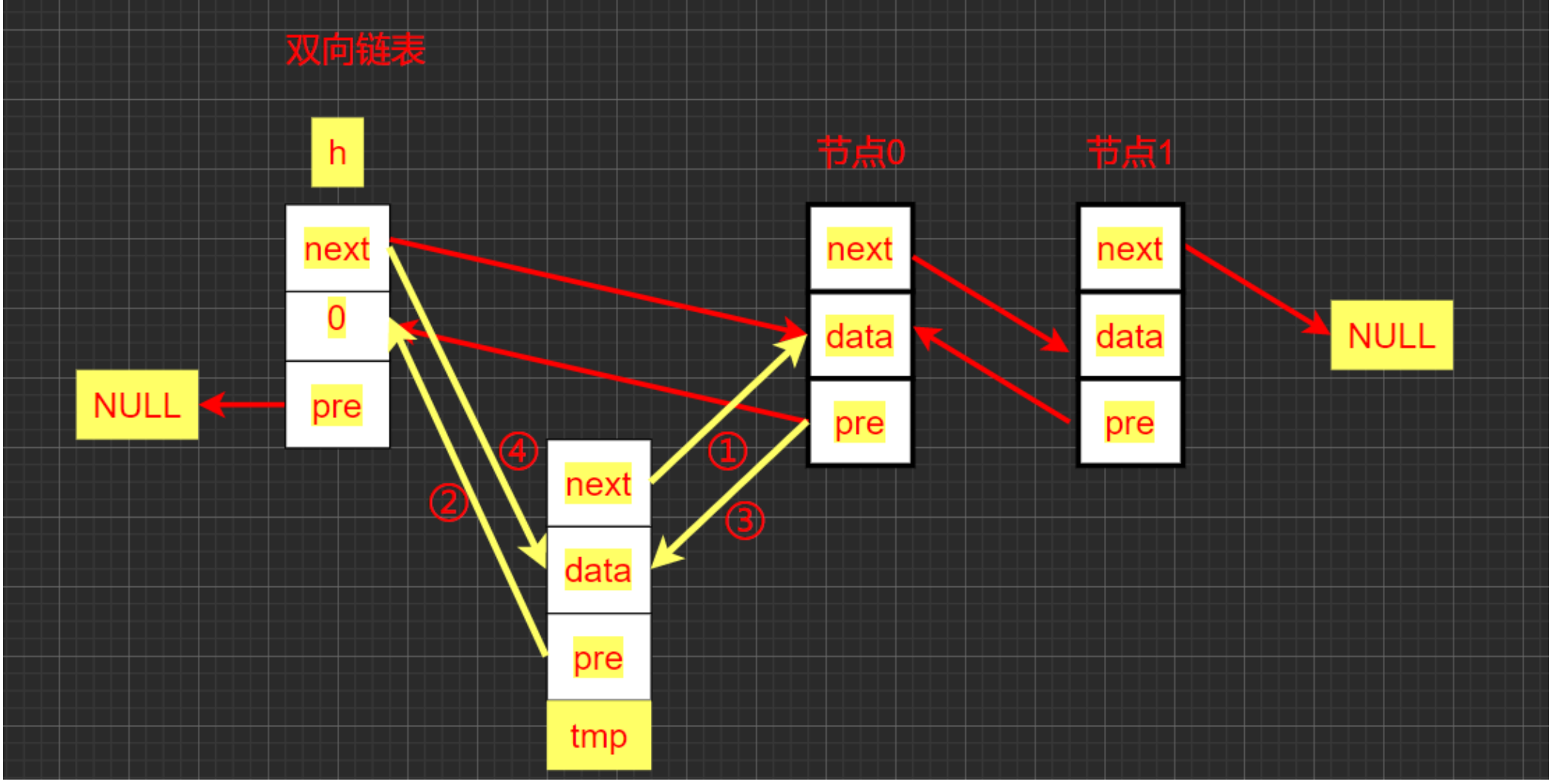
情况1：（双链表不为空链表）

tmp->next = h->next;

tmp->pre = h;

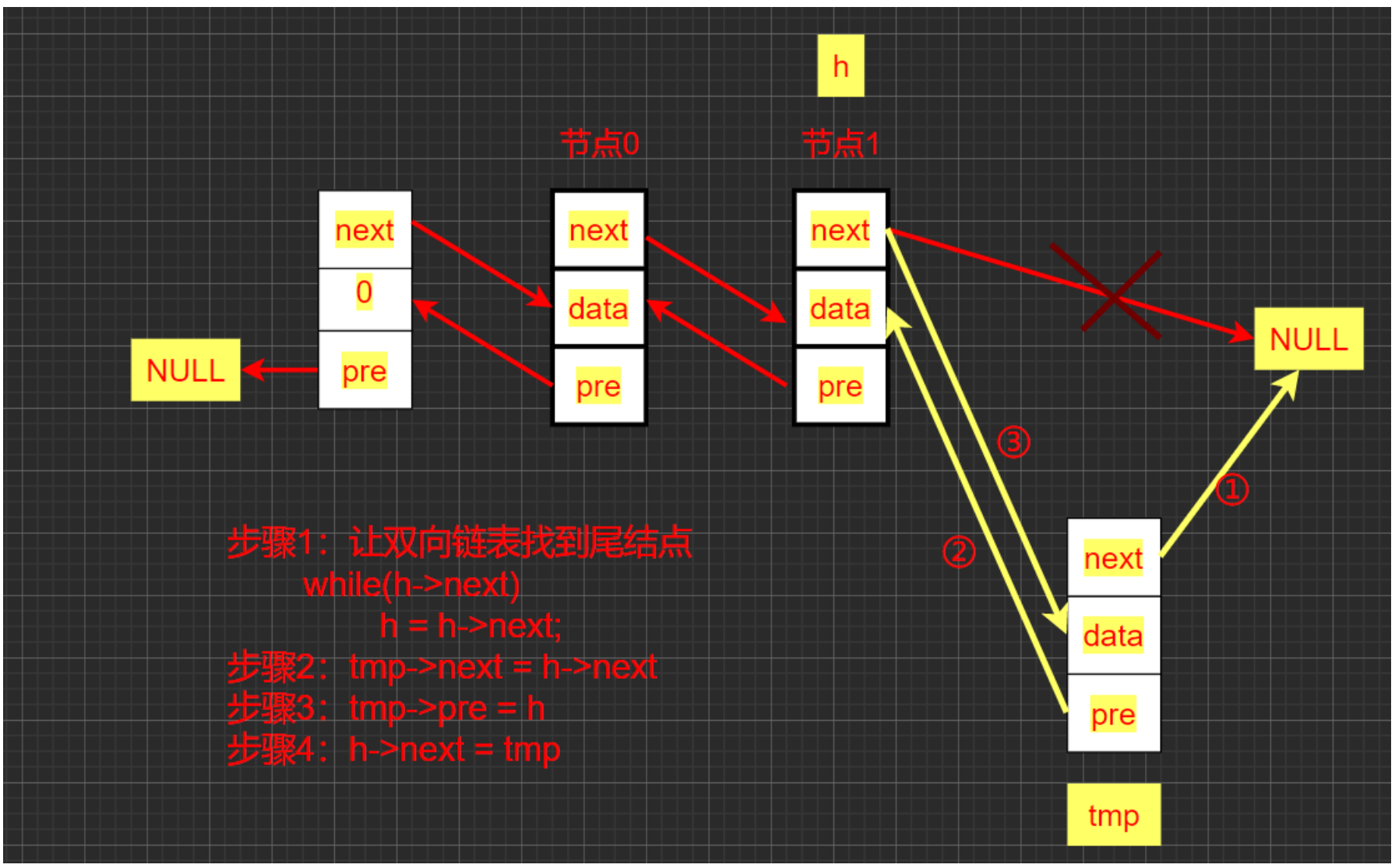
h->next->pre = tmp;

h->next = tmp;




```
int DPListInsertHead(DPlist_t* h, datatype data)
{
    DPlist_t* tmp;
    tmp = (DPlist_t*)malloc(sizeof(*tmp));
    if (tmp == NULL) {
        printf("%s malloc memory error\n", __func__);
        return -1;
    }
    tmp->data = data;

    tmp->next = h->next;
    tmp->pre = h;
    if (h->next != NULL)
        h->next->pre = tmp;
    h->next = tmp;

    return 0;
}
```

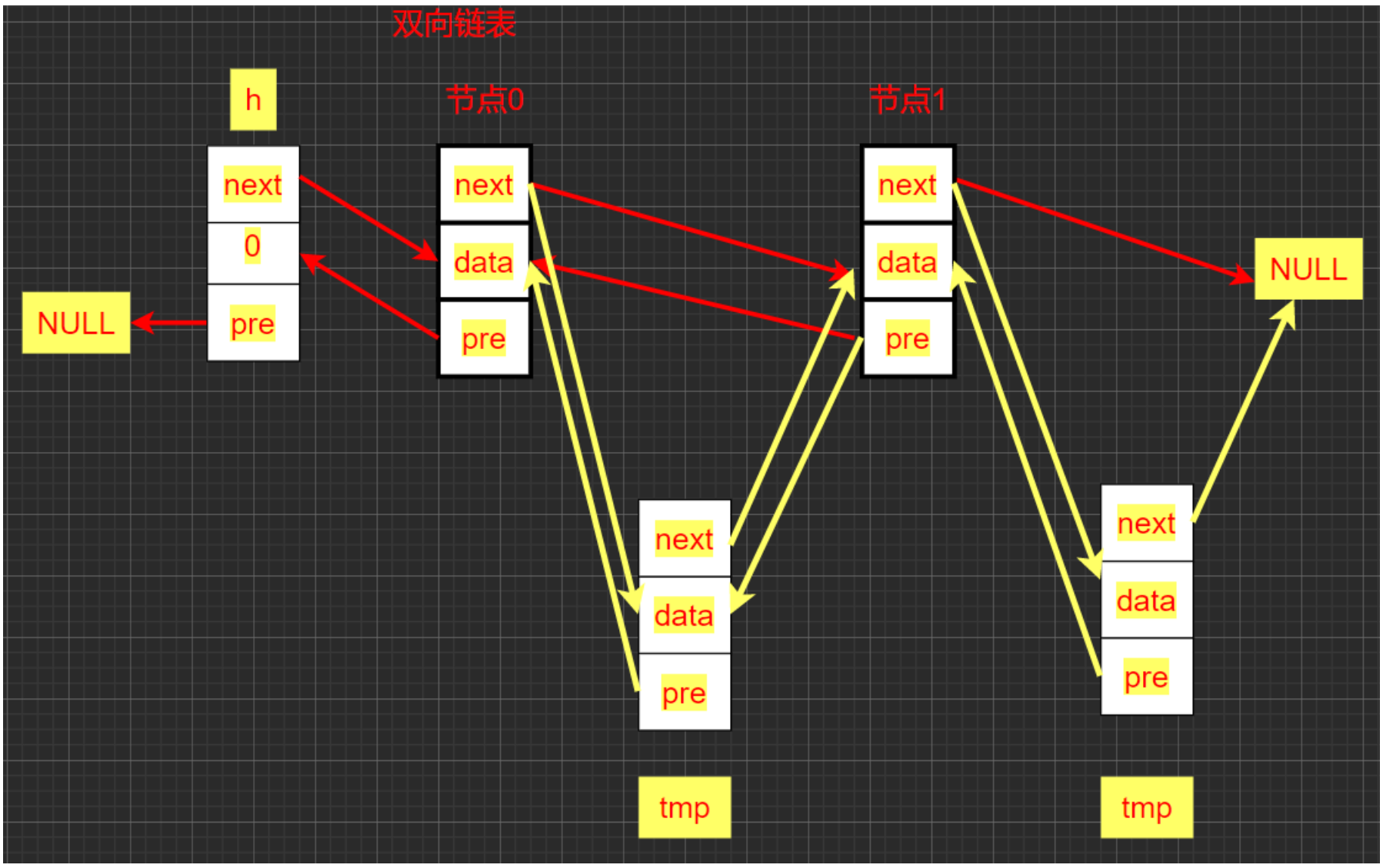
1.3.3双向链表尾插




```
int DPListInsertTail(DPlist_t* h, datatype data)
{
    DPlist_t* tmp;
```

```
    // 1.分配tmp节点，将data存入
    tmp = (DPlist_t*)malloc(sizeof(*tmp));
    if (tmp == NULL) {
        printf("%s malloc memory error\n", __func__);
        return -1;
    }
    tmp->data = data;
    // 2.让h走到尾节点
    while (h->next)
        h = h->next;
    // 3.节点插入
    tmp->next = h->next;
    tmp->pre = h;
    h->next = tmp;
    return 0;
}
```

1.3.4双向链表位置插




```
int DPListInsertByPos(DPlist_t* h, int pos, datatype data)
{
    if (pos < 0) {
        printf("%s pos left error\n", __func__);
        return -1;
    }
    while (h) {
        if (pos != 0) {
            h = h->next;
            pos--;
        } else {
            // 找到插入的位置
            DPlist_t* tmp;
            tmp = (DPlist_t*)malloc(sizeof(*tmp));
            if (tmp == NULL) {
                printf("%s malloc memory error\n", __func__);
                return -1;
            }
            tmp->data = data;

            tmp->next = h->next;
            tmp->pre = h;
            if(h->next != NULL)
                tmp->next->pre = tmp;
            h->next = tmp;
            return 0;
        }
    }
    printf("%s pos right error\n", __func__);
    return -1;
}
```

1.3.5双向链表的遍历

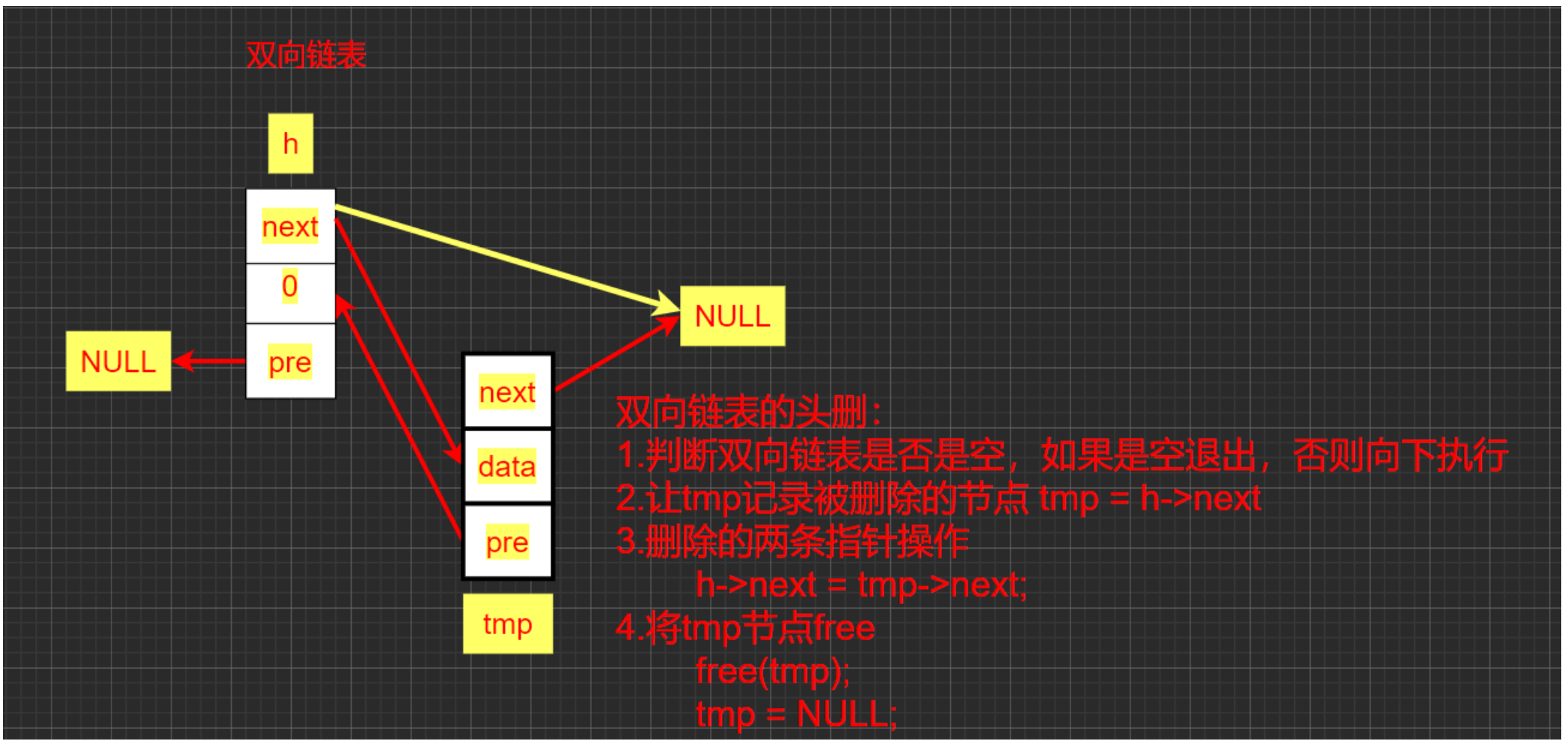
```c
void DPListShow(DPlist_t* h)
{
    printf("双向链表正向遍历：");
    while (h->next) {
        printf("-%d", h->next->data);
        h = h->next;
    }
    printf("-\n");
    printf("双向链表逆向遍历：");
    while (h->pre) {
        printf("-%d", h->data);
        h = h->pre;
    }
    printf("-\n");
}
```

1.3.6双向链表判空

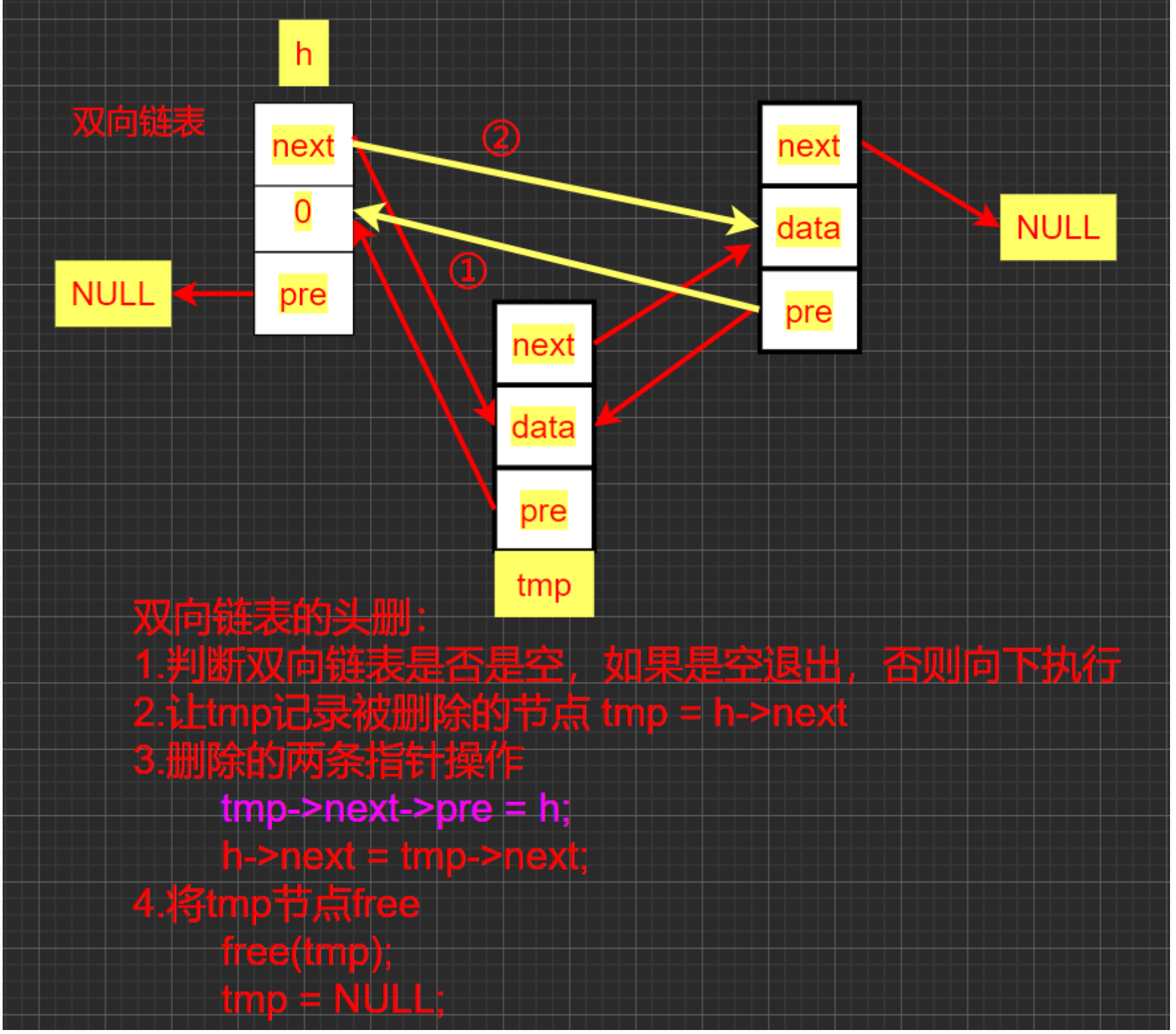
```c
int DPListIsEmpty(DPlist_t* h)
{
    return h->next == NULL ? 1 : 0;
}
```

1.3.7双向链表的头删

情况1：（双链表中只有一个节点）

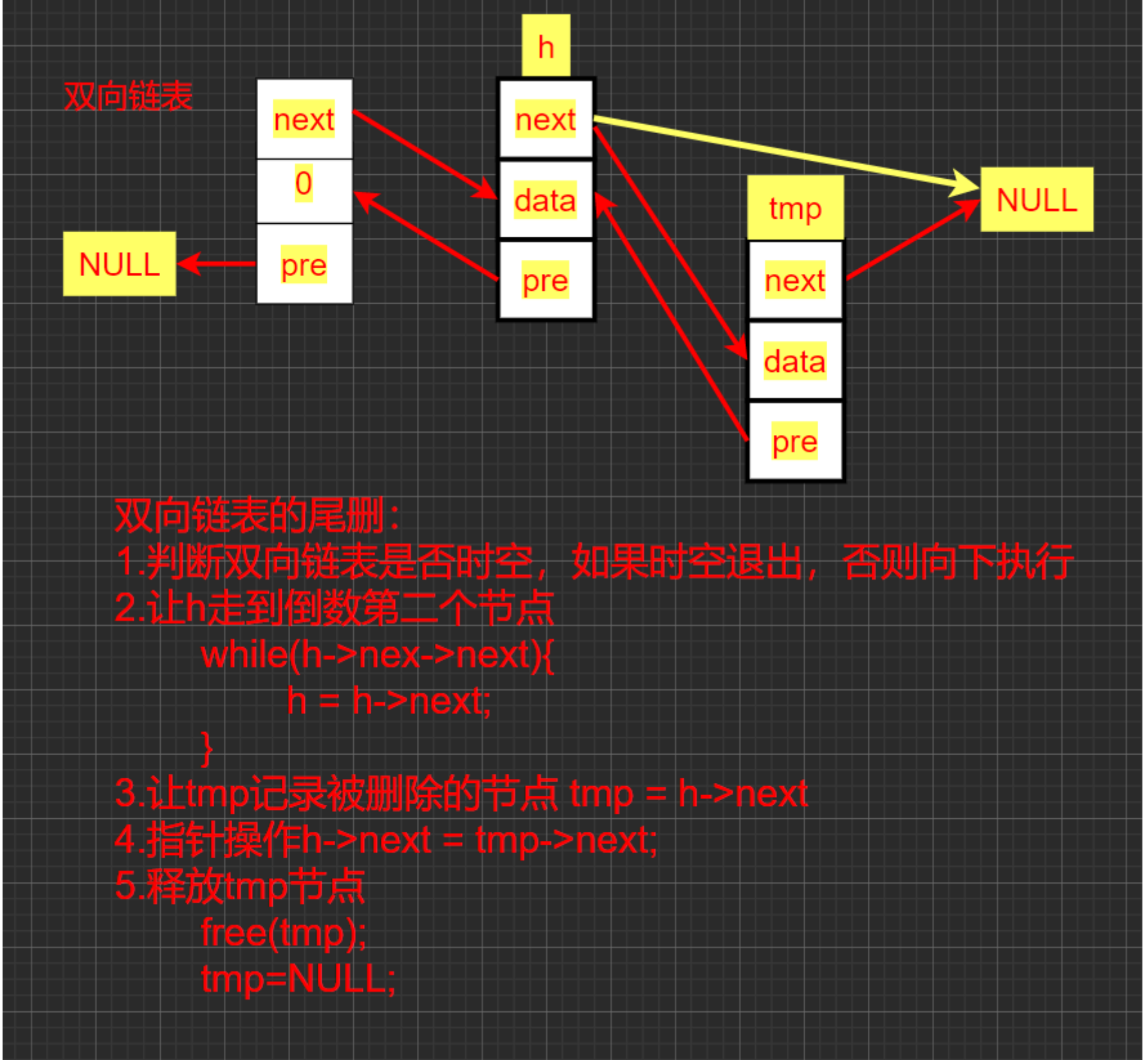


情况2：（双链表中有多个节点）



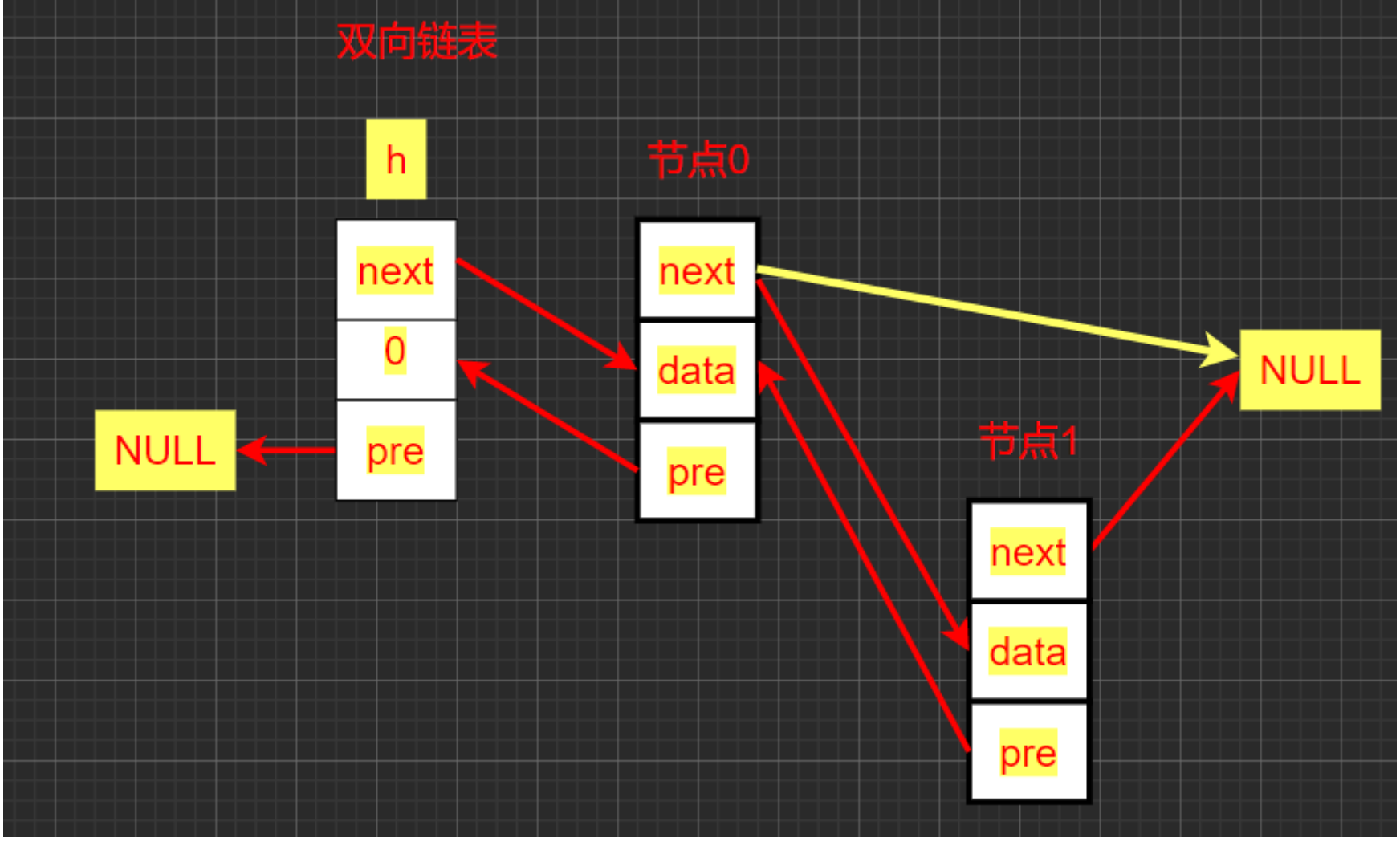

```c
datatype DPListDeleteHead(DPlist_t* h)
{
    datatype data;
    DPlist_t* tmp;
    if (DPListIsEmpty(h)) {
        printf("%s list empty\n", __func__);
        return (datatype)-1;
    }

    tmp = h->next;
    if (tmp->next != NULL)
        tmp->next->pre = h;
    h->next = tmp->next;

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

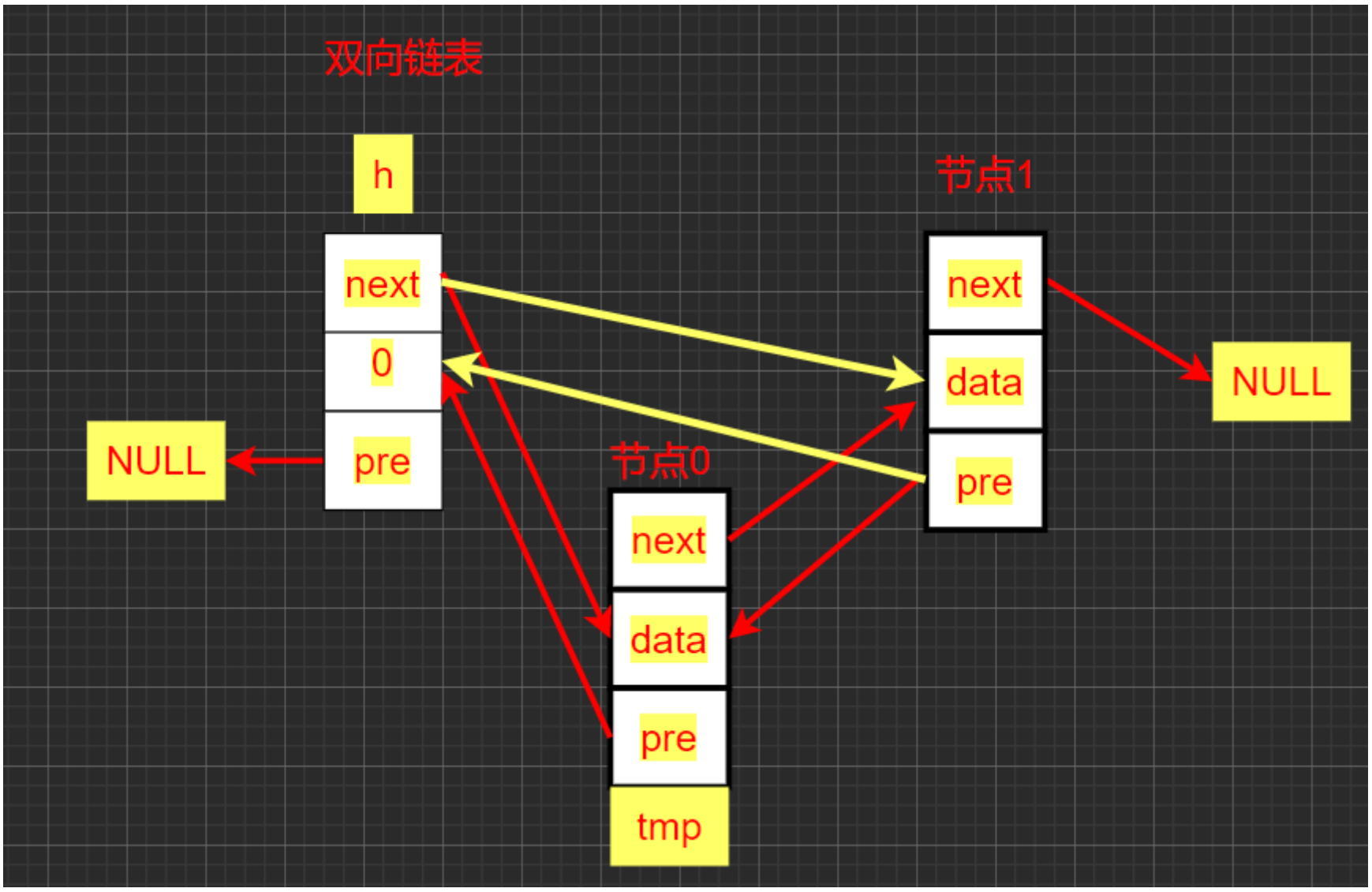
1.3.8双向链表的尾删




```c
datatype DPListDeleteTail(DPlist_t* h)
{
    datatype data;
    DPlist_t* tmp;

    if (DPListIsEmpty(h)) {
        printf("%s list empty\n", __func__);
        return (datatype)-1;
    }

    while (h->next->next)
        h = h->next;

    tmp = h->next;

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

    return data;
}
```

1.3.9双向链表的位置删






```c
datatype DPListDeleteByPos(DPlist_t* h, int pos)
{
    if (pos < 0) {
        printf("%s pos left error\n", __func__);
        return (datatype)-1;
    }
    while (h->next) {
        if (pos != 0) {
            h = h->next;
            pos--;
        } else {
            // 找到删除的位置
            DPlist_t* tmp;
            datatype data;
            tmp = h->next;

            if (tmp->next != NULL)
                tmp->next->pre = h;
            h->next = tmp->next;

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

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

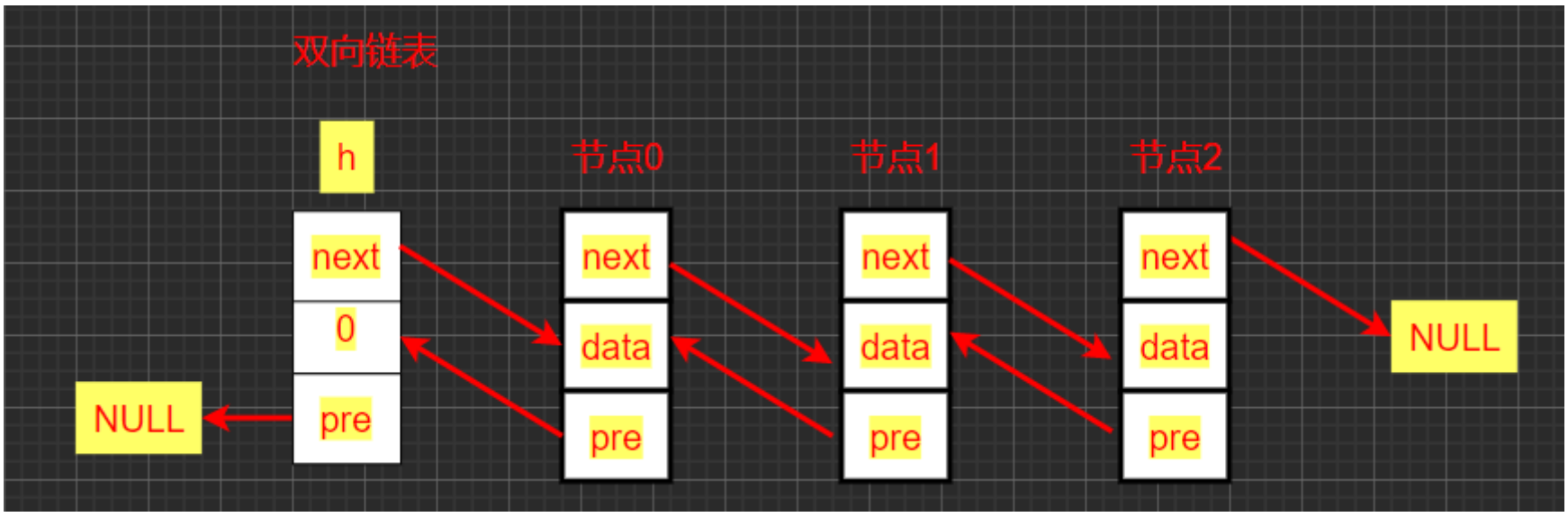
1.3.10双向链表的查询

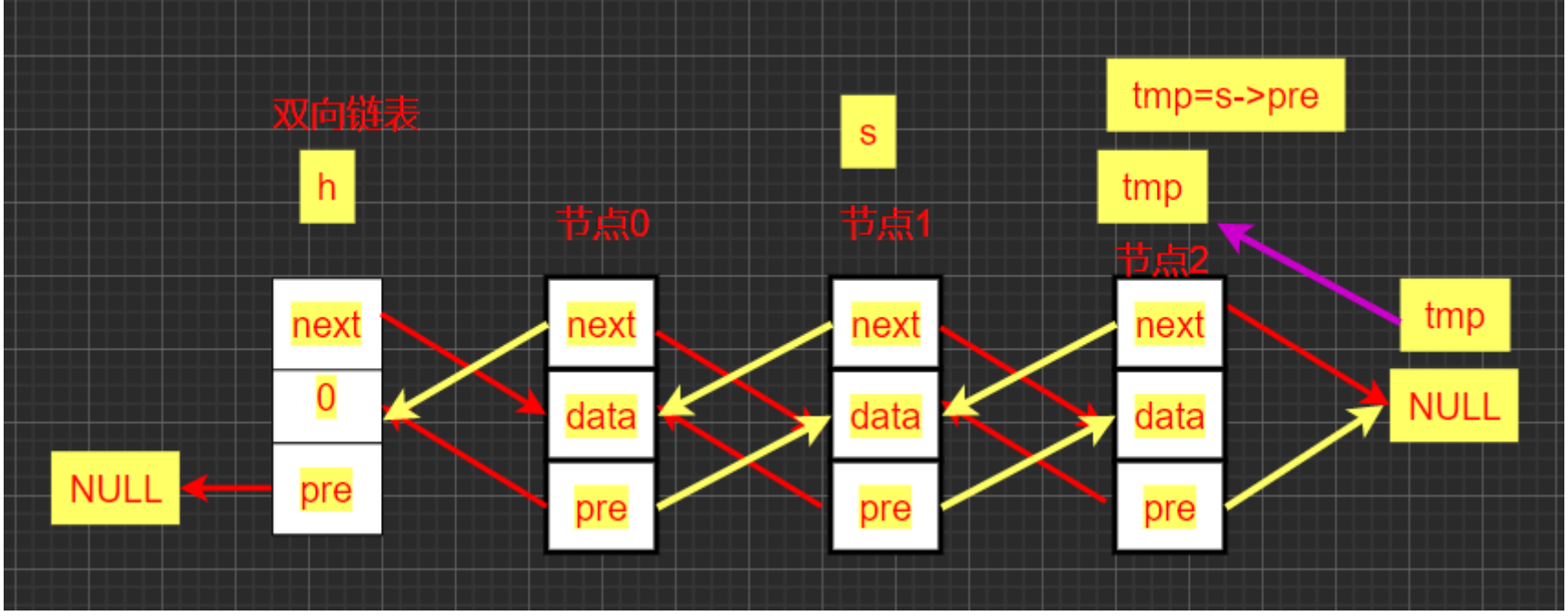
```
datatype DPListCheckDataByPos(DPlist_t* h, int pos)
{
    if (pos < 0) {
        printf("%s pos left error\n", __func__);
        return (datatype)-1;
    }
    while (h->next) {
        if (pos != 0) {
            h = h->next;
            pos--;
        } else {
            // 找到查询的位置
            return h->next->data;
        }
    }
    printf("%s pos right error\n", __func__);
    return (datatype)-1;
}
```

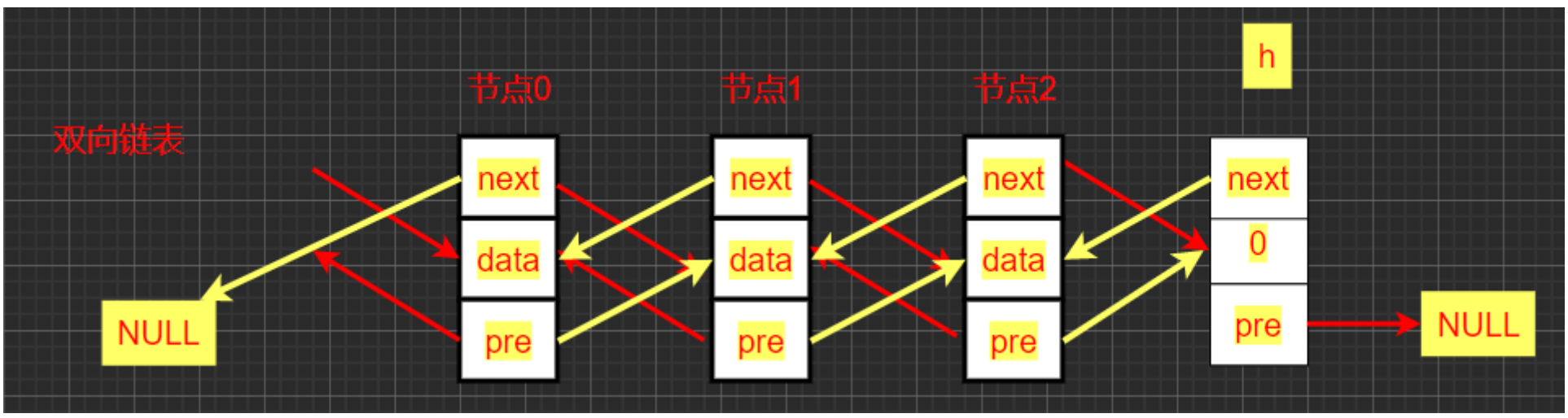
1.3.11双向链表的更新

```
int DPListUpdateDataByPos(DPlist_t* h, int pos, datatype data)
{
    if (pos < 0) {
        printf("%s pos left error\n", __func__);
        return -1;
    }
    while (h->next) {
        if (pos != 0) {
            h = h->next;
            pos--;
        } else {
            // 找到更新的位置
            h->next->data = data;
            return 0;
        }
    }
    printf("%s pos right error\n", __func__);
    return -1;
}
```

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

```c
#include "DPlist.h"

DPlist_t* DPListCreate(void)
{
    DPlist_t* h;
    h = (DPlist_t*)malloc(sizeof(*h));
    if (h == NULL) {
        printf("%s malloc memory error\n", __func__);
        return NULL;
    }
    h->data = (datatype)0;
    h->next = NULL;
    h->pre = NULL;

    return h;
}
int DPListInsertHead(DPlist_t* h, datatype data)
{
    DPlist_t* tmp;
    tmp = (DPlist_t*)malloc(sizeof(*tmp));
    if (tmp == NULL) {
        printf("%s malloc memory error\n", __func__);
        return -1;
    }
    tmp->data = data;

    tmp->next = h->next;
    tmp->pre = h;
    if (h->next != NULL)
        h->next->pre = tmp;
    h->next = tmp;

    return 0;
}
int DPListInsertTail(DPlist_t* h, datatype data)
{
    DPlist_t* tmp;
    // 1.分配tmp节点，将data存入
    tmp = (DPlist_t*)malloc(sizeof(*tmp));
    if (tmp == NULL) {
        printf("%s malloc memory error\n", __func__);
        return -1;
    }
    tmp->data = data;
    // 2.让h走到尾节点
    while (h->next)
        h = h->next;
    // 3.节点插入
    tmp->next = h->next;
    tmp->pre = h;
    h->next = tmp;
    return 0;
}
int DPListInsertByPos(DPlist_t* h, int pos, datatype data)
{
    if (pos < 0) {
        printf("%s pos left error\n", __func__);
        return -1;
    }
    while (h) {
        if (pos != 0) {
            h = h->next;
            pos--;
        } else {
            // 找到插入的位置
            DPlist_t* tmp;
            tmp = (DPlist_t*)malloc(sizeof(*tmp));
            if (tmp == NULL) {
                printf("%s malloc memory error\n", __func__);
                return -1;
            }
            tmp->data = data;

            tmp->next = h->next;
            tmp->pre = h;
            if (h->next != NULL)
                tmp->next->pre = tmp;
            h->next = tmp;
            return 0;
        }
    }
    printf("%s pos right error\n", __func__);
```

```
    return -1;
}
void DPListShow(DPlist_t* h)
{
    printf("双向链表正向遍历：");
    while (h->next) {
        printf("-%d", h->next->data);
        h = h->next;
    }
    printf("-\n");
    printf("双向链表逆向遍历：");
    while (h->pre) {
        printf("-%d", h->data);
        h = h->pre;
    }
    printf("-\n");
}
int DPListIsEmpty(DPlist_t* h)
{
    return h->next == NULL ? 1 : 0;
}
datatype DPListDeleteHead(DPlist_t* h)
{
    datatype data;
    DPlist_t* tmp;
    if (DPListIsEmpty(h)) {
        printf("%s list empty\n", __func__);
        return (datatype)-1;
    }

    tmp = h->next;
    if (tmp->next != NULL)
        tmp->next->pre = h;
    h->next = tmp->next;

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

    return data;
}
datatype DPListDeleteTail(DPlist_t* h)
{
    datatype data;
    DPlist_t* tmp;

    if (DPListIsEmpty(h)) {
        printf("%s list empty\n", __func__);
        return (datatype)-1;
    }

    while (h->next->next)
        h = h->next;

    tmp = h->next;

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

    return data;
}

datatype DPListDeleteByPos(DPlist_t* h, int pos)
{
    if (pos < 0) {
        printf("%s pos left error\n", __func__);
        return (datatype)-1;
    }
    while (h->next) {
        if (pos != 0) {
            h = h->next;
            pos--;
        } else {
            // 找到删除的位置
            DPlist_t* tmp;
            datatype data;
            tmp = h->next;

```

```
            if (tmp->next != NULL)
                tmp->next->pre = h;
            h->next = tmp->next;

            data = tmp->data;
            if (tmp != NULL) {
                free(tmp);
                tmp = NULL;
            }
            return data;
        }
    }
    printf("%s pos right error\n", __func__);
    return (datatype)-1;
}

datatype DPListCheckDataByPos(DPlist_t* h, int pos)
{
    if (pos < 0) {
        printf("%s pos left error\n", __func__);
        return (datatype)-1;
    }
    while (h->next) {
        if (pos != 0) {
            h = h->next;
            pos--;
        } else {
            // 找到查询的位置
            return h->next->data;
        }
    }
    printf("%s pos right error\n", __func__);
    return (datatype)-1;
}
int DPListUpdateDataByPos(DPlist_t* h, int pos, datatype data)
{
    if (pos < 0) {
        printf("%s pos left error\n", __func__);
        return -1;
    }
    while (h->next) {
        if (pos != 0) {
            h = h->next;
            pos--;
        } else {
            // 找到更新的位置
            h->next->data = data;
            return 0;
        }
    }
    printf("%s pos right error\n", __func__);
    return -1;
}

void DPListReverse(DPlist_t* h)
{
    DPlist_t *tmp, *s;
    if (DPListIsEmpty(h))
        return;
    tmp = h->next;
    while (tmp) {
        s = tmp->pre;
        tmp->pre = tmp->next;
        tmp->next = s;
        tmp = tmp->pre;
    }
    tmp = s->pre;

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

main.c

```
#include "DPlist.h"
int main(int argc, const char* argv[])
{
    DPlist_t* h;

    h = DPListCreate();
    if (h == NULL)
        return -1;
```

```
    DPListInsertHead(h, 123);
    DPListInsertHead(h, 567);
    DPListInsertHead(h, 888);
    DPListInsertHead(h, 4390);
    DPListShow(h);
    // DPListInsertTail(h, 11);
    // DPListInsertTail(h, 22);
    // DPListShow(h);
    // DPListInsertByPos(h, 1, 777);
    // DPListShow(h);

    // DPListDeleteHead(h);
    // DPListShow(h);
    // DPListDeleteHead(h);
    // DPListDeleteTail(h);
    // DPListShow(h);
    // DPListDeleteByPos(h,1);
    // DPListShow(h);

    // printf("check data = %d\n",DPListCheckDataByPos(h,1));
    // DPListUpdateDataByPos(h,1,666);
    // DPListShow(h);
    puts("----------------------------------------");
    DPListReverse(h);
    DPListShow(h);
    return 0;
}
```