要想改变结构体里面的内容，只需要结构体指针，如果不需要对结构体指针进行操作改变，则不需要传递二级指针

将二级指针init改为一级指针init的方法

**链表一共有八种结构**

\*（1）

（2）

（3）

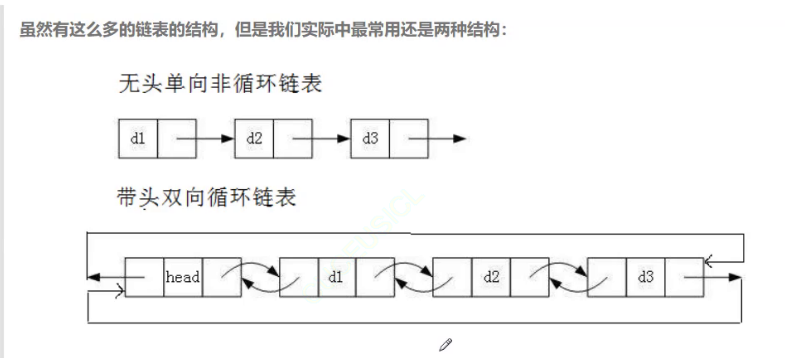
（4）

（5）

（6）

（7）

（8）



**带头循环双链表的结构**

#

**创建空间**

LTNode\* BuyLTNode(LTDataType x)

{

LTNode \* newnode = (LTNode \*)malloc(sizeof(LTNode));

if(newnode == NULL)

{

perror("malloc fail\n");

return NULL;

}

}

**带头循环双链表的初始化**

LTNode \* LTInit(void)

{

LTNode \* phead = BuyLTNode(-1);

phead->next = phead;//尾指针指向自己

phead->prev = phead;//头指针也指向自己

return phead;//主函数中需要有一个指针来接收这个返回值

//这样做可以不用传参传二级指针

}

**带头循环双链表的尾插**

void LTPushBack(LTNode \* phead,LTData x)

{

assert(phead);//头指针不能为空

LTNode \* tail = phead->prev;//tail是已经存在的最后一个节点，尾结点是由头结点的头指针所指向

LTNode \* newnode = BuyLTNode(x);//在内存中申请一块空间用在存放数据x

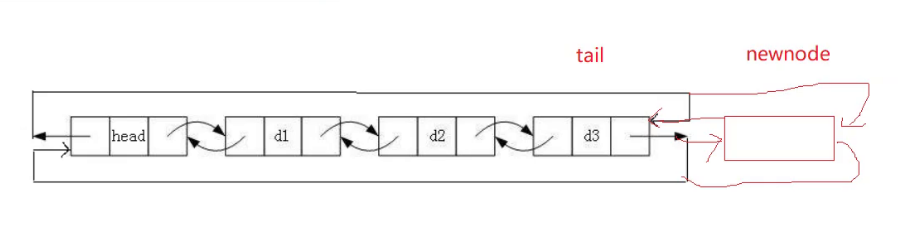
tail->next = newnode;//尾结点的尾指针指向新结点

newnode->prev = tail;//新节点的头指针指向尾结点

newnode->next = phead;//新节点的尾指针指向头结点

phead->prev = newnode;//头结点的头指针指向新节点

}



**带头循环双链表的头插**

void LTPushFront(LTNode \* phead,LTData x)

{

aassert(phead);

LTNode \* newnode = BuyLTNode(x);

newnode->next = phead->next;//新节点尾指针先指向头结点（如果链表为空则指向首节点）

newnode->prev = phead;//新节点头指针指向首结点（哨兵位）

phead->next = newnode;//首结点（哨兵位）的尾指针指向新节点

newnode->next->prev = newnode;//原来的头结点（如果为空链表则为首节点）的头指针指向新的节点

}

**判断是否为空**

#include<stdbool.h>

bool LTEmpty(LTNode \* phead)

{

assert(phead);

return (phead->next == phead);

}

**带头循环双链表头删**

//只用一个指针的写法(可读性较差)

void LTPopFront(LTNode \* phead)

{

assert(phead);//首节点不为空

assert(!LTEmpty(phead));//链表不为空

phead->next = phead->next->next;//首节点尾指针指向第二个节点

free(phead->next->prev);//释放掉第二个结点头指针指向的节点

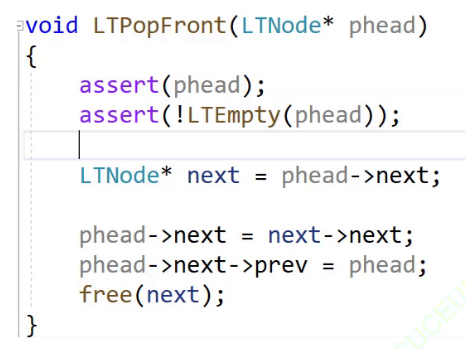
phead->next->prev = phead;//之前第二个结点的头结点指向首节点

}

可读性较好版本：



只带两个指针的写法：（没问题但是容易坑）



**断言条件要分开写，不然出错了不知道是哪一条还要去查找**

如：

assert(A && B)

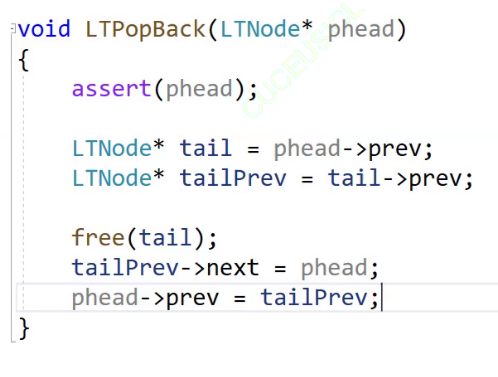
要写成->

assert(A);

assert(B);

**带头循环双链表尾删**

小问题代码，不能删除完，必须留一个首结点，如果链表已经为空还继续删除将会出问题



#include<stdbool.h>

void LTPopBack(LTNode \* phead)

{

assert(phead);//首节点不为空

assert(!LTEmpty(phead));//链表不为空

LTNode \* tail = phead->prev;//找到尾结点

LTNode \* tailPrev = tail->prev;//找到尾结点的前驱结点

free(tail);//释放尾结点

tailPrev->next = phead;//新的尾结点的尾指针指向首节点

phead->prev = tailPrev;//首节点的头指针指向新的尾结点

}

**带头循环双链表打印**

void LTPrint(LTNode \* phead)

{

assert(phead);

printf("leader<==>");

LTNode \*cur = phead->next;

while(cur != phead)

{

printf("%d<==>",cur->data);

cur = cur->next;

}

printf("\n");

}

**带头循环双链表查找**

LTNode \* LTFind(LTNode \* phead,LTData x/\*该结点存放的x\*/)

{

assert(phead);

LTNode \* cur = phead->next;

while(cur->data == x)

{

if(cur->data == x)

{

return cur;

}

cur = cur->next;

}

return NULL;

}

**在pos之前插入**

void LTInsert(LTNode \* pos,LTData x)//\*\*\*加入基础

{

assert(pos);

LTNode \*prev = pos->prev;

LTNode \*newnode = BuyLTNode(x);

//prev newnode pos

prev->next = newnode;//前驱结点的尾指针指向新节点

newnode->pre = prev;//新节点的头指针指向前驱结点

newnode->next = pos;//新节点尾指针指向pos

pos->prev = newnode;//pos的头结点指向新节点

}

该用指针就用指针，地方够用，可读性也要顾及

**删除pos的值**

void LTNode(LTNode \* pos)//\*\*\*删除基础

{

assert(pos);

LTNode \* posPrev = pos->prev;

LTNode \* posNext = pos->next;

posPrev->next = posNext;

posNext->prev = posPrev;

free(pos);

}

然后用两个基本函数实现简洁的头插尾差，头删尾删

////////////

//头插

void LTNodePushFront(LTNode \*phead,LTData x)

{

assert(head);

LTInsert(phead->next,x);//复用

}

//尾差

void LTNodePushBack(LTNode \*phead,LTData x)

{

assert(head);

LTInsert(phead,x);//复用

}

//头删

void LTPopFront(LTNode \* phead)

{

assert(phead);

assert(!LTEmpty(phead));

LTErase(phead->next);

}

//尾删

void LTPopBack(LTNode \*phead)

{

assert(phead);

assert(!LTEmpty(head));

LTEErase(phead->prev);

}

**带头循环双链表删除**

void LTDestroy(LTNode \*phead)

{

assert(phead);

LTNode \* cur = phead->next;

while(cur != phead)

{

LTNode \*next = cur->next;

free(cur);

cur = next;

}

free(phead);

}