大家好，我是蓝蓝，这是我们一期学习专题算法的第25天。day25/45

蓝蓝B站首页：蓝蓝希望你上岸呀B站首页

蓝蓝公众号：算法训练营9分计划

对应的公开视频讲解：系列视频

1、知识点及难度

- 知识点：顺序表
- 难度：中等

2、题目描述

注意：对于初试只需要写关键代码即可

> 题目描述：
> 设 L 为带头结点的单链表，编写算法实现从尾到头反向输出每个结点的值

> 思路

大家好，这是我们准备计算机考研算法的第27天。前面两天分别掌握学习了单链表创建的尾插法和头插法，在后面的几天学习中都会用到类似的思想。

今天的这个题目其实就是头插法的巩固。我们一起看题。

其实这个题目不是很清晰，隐藏了几点。

- 有一个采用尾插法建立的链表
- 现在需要对这个链表，采用头插法的方式插入到另一个链表，和后面的就地逆置题不一样
- 然后进行一个输出

这里举了一个例子。链表L和链表Reverse，每次从链表L中取出一个元素，然后头插到Reverse中，实际上就得到了我们想要的结果。

采用的步骤

- 创建一个Reverse结点S
- 第二步取出L指向结点的值并赋值给刚才创建的结点S
- 将结点L头插到链表Reverse
- 循环

【思路】

1. 尾插法建立单链表 L；L；
2. 将上述单链表中的元素按从头到尾的顺序，使用头插法新建一个链表 reverse；
3. 打印输出 reverse 中的元素。

```
// 头插法建立单链表，反向输出原单链表的元素值
bool list_head_insert(LinkList L, LinkList &reverse) {
    if (!InitList(reverse)) {
```

```
        return false;
    }
    L = L->next; // 使得 L 单链表的头指针指向其第一个元素
    LNode *s;
    while (L != NULL) {
        s = (LNode *) malloc(sizeof(LNode));
        s->data = L->data;
        s->next = reverse->next;
        reverse->next = s;
        L = L->next;
    }
    return true;
}
```

> 实现

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

typedef int ElemType;

typedef struct LNode {
    ElemType data; // 数据域
    struct LNode *next; // 指针域
} LNode, *LinkList;

// 初始化单链表
bool InitList(LinkList &L) {
    L = (LNode *) malloc(sizeof(LNode));
    if (L == NULL) { // 内存不足，分配失败
        return false;
    }
    L->next = NULL;
    return true;
}

// 打印单链表
void PrintList(LinkList L) {
    L = L->next;
    while (L != NULL) {
        printf("%3d", L->data);
        L = L->next;
    }
    printf("\n");
}

// 尾插法建立单链表
bool list_tail_insert(LinkList &L) {
    if (!InitList(L)) { // 链表初始化
```

```
        return false;
    }
    ElemType x;
    LNode *s, *r = L; // 声明头指针和尾指针
    scanf("%d", &x);
    while (x != 999) {
        s = (LNode *) malloc(sizeof(LNode));
        s->data = x;
        r->next = s;
        r = s; // r 指向新的尾结点
        scanf("%d", &x);
    }
    r->next = NULL; // 尾结点指针置空
    return true;
}

// 头插法建立单链表，反向输出原单链表的元素值
bool list_head_insert(LinkList L, LinkList &reverse) {
    if (!InitList(reverse)) {
        return false;
    }
    L = L->next; // 使得 L 单链表的头指针指向其第一个元素
    LNode *s;
    while (L != NULL) {
        s = (LNode *) malloc(sizeof(LNode));
        s->data = L->data;
        s->next = reverse->next;
        reverse->next = s;
        L = L->next;
    }
    return true;
}

// 参考答案 -- 递归思想
void reverse_print(LinkList L) {
    if (L->next != NULL) {
        reverse_print(L->next); // 递归
    }
    if (L != NULL) {
        printf("%3d", L->data);
    }
}

void reverse_ignore_head(LinkList L) { // 不打印头结点中的内容
    if (L->next != NULL) {
        reverse_print(L->next);
    }
}

int main() {

    LinkList L;
    LinkList reverse;

    bool ret;

    // 尾插法建立单链表
    ret = list_tail_insert(L);
```

```
    if (ret) {
        printf("origin list:");
        PrintList(L);
    } else {
        printf("list_tail_insert failed!\n");
    }

    // 头插法建立单链表，反向输出原单链表的元素
    ret = list_head_insert(L, reverse);
    if (ret) {
        printf("through head_insert:");
        PrintList(reverse);
    } else {
        printf("print reversed list failed!\n");
    }

    // 通过递归的方法反向输出【参考答案】
    printf("through recursion:");
    reverse_ignore_head(L);

    return 0;
}
```