单链表反序：

例：程序C++ Algorithm\qq\_test1

// 链表反序

#include <iostream>

using std::cin;

using std::cout;

using std::endl;

typedef struct List

{

int number;

struct List\* next;

} List;

List\* CreateList(List\* head);

void PrintList(List\* head);

List\* ReverseList(List\* head);

void ListTest(List\* head);

int main(int argc, char\* argv[])

{

List\* head = NULL;

head = CreateList(head);

ListTest(head);

PrintList(head);

cout << "链表反序" << endl;

head = ReverseList(head);

// ReverseList(head); // 这样操作是不会将链表反序，因为在函数中修改指针本身是没有用的

PrintList(head);

return 0;

}

void ListTest(List\* head)

{

head->number = 1; // 会修改函数外部实参中的值

head = NULL; // 不会影响函数外部实参的值

}

// 创建链表

List\* CreateList(List\* head)

{

List\* list1\_ptr = NULL;

List\* list2\_ptr = NULL;

list1\_ptr = new List;

// list2\_ptr = new List;

memset(list1\_ptr, 0, sizeof(List));

cin >> list1\_ptr->number;

// 如果每次新节点的元素number大于0，则增加节点

// 初始化状态下head为NULL，第一次循环，head==NULL，将head

// 指向节点list1\_ptr，并将list2\_ptr指向list1\_ptr，这样list2\_ptr

// 与head指向同一个节点；第二次循环，将list2\_ptr->next指向新产生

// 的节点list1\_ptr，这样相当于将head->next指向了新的list1\_ptr，然后

// 将list2\_ptr指向list1\_ptr；第三次循环，将list2\_ptr->next指向新产生

// 的节点list1\_ptr，实际是将第二个节点的next指针指向了新产生的list1\_ptr

// 这样就构成了链表

while (list1\_ptr->number > 0)

{

if (head == NULL)

{

head = list1\_ptr;

}

else

{

list2\_ptr->next = list1\_ptr;

}

list2\_ptr = list1\_ptr;

list1\_ptr = new List; // 将list1\_ptr当成每次添加的新节点

memset(list1\_ptr, 0, sizeof(List));

cin >> list1\_ptr->number;

list1\_ptr->next = NULL;

}

return head;

}

void PrintList(List\* head)

{

if (head == NULL)

{

cout << "链表为空" << endl;

}

else

{

while (head != NULL)

{

cout << head->number << endl;

// 在函数中修改指针本身，不会影响函数外部的实参指针

head = head->next;

}

}

}

// 链表反序

List\* ReverseList(List\* head)

{

if (head == NULL)

{

cout << "链表为空" << endl;

return head;

}

List\* p1 = NULL;

List\* p2 = NULL;

List\* p3 = NULL;

p1 = head;

p2 = p1->next;

// p1：指向原表中当前节点的上级节点

// p2：指向原表中当前节点

// p3：指向原表中当前节点的后继节点

for (; p2 != NULL;)

{

// 将当前节点p2的下一个节点存储在p3中

p3 = p2->next;

// 将当前节点p2的原上级节点p1当成后继节点

p2->next = p1;

// 将p1指向当前节点

p1 = p2;

// 将p2指向原表中的下一节点

p2 = p3;

}

// 原链表的新节点为反序后新链表的尾节点，所以head->next指向NULL

head->next = NULL;

// p1会最终指向原链表的尾节点，即新链表的头节点，所以将head指向p1

head = p1;

return head;

}

