#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "s.h"

//获取节点数目(包括了头结点)

unsigned int GetListLength(link \* head)

{

unsigned int n= 0;

link \* p = head;

if(head == NULL) return 0;

while(NULL!=p)

{

n++;

p=p->next;

}

return n;

}

//单链表反转返回参数版

//1头插法

link \* ReverseList\_by\_insert\_head(link \* head)

{

link \* pele = NULL;

link \* p = head->next;

if(p == NULL || p->next == NULL) return head;

head->next=NULL;

while(NULL!=p)

{

pele=p;

p=p->next;

//头插入

pele->next=head->next;

head->next=pele;

}

return head;

}

//2递归法

link \* ReverseList\_by\_recursion(link \* head)

{

if(head == NULL || head->next == NULL) return head;

link \* new=ReverseList\_by\_recursion(head->next);

if(0!=head->ch)

{

head->next->next=head;//翻转链表的指向!

head->next=NULL;//记得赋值NULL，防止链表错乱

return new;//新链表头永远指向的是原链表的链尾

}

else

{

head->next=new;

return head;

}

}

// 获取单链表中间结点，若链表长度为n(n>0)，则返回第n/2+1个节点

link \* GetMiddleNode(link \* head)

{

if(head == NULL || head->next == NULL) return head; // 链表为空或只有一个节点，返回头指针

link \* p = head;

link \* pMed =head;

while(NULL!=p->next) // p指针每次走两步，pMed每次走一步

{

p = p->next;

pMed = pMed->next;

if(NULL !=p->next )

p = p->next;

}

return pMed; // 后面的指针所指节点即为中间节点

}

//已知两单链表各自有序，合并后依然有序---类似于归并排序

link \* MergeSortedList(link \* head1, link \* head2)

{

link \*p1= head1->next;

link \*p2= head2->next;

if(p1== NULL)

return head2;

if(p2 == NULL)

return head1;

//比较第一个有效节点

link \* headMerged=(link\*)malloc(sizeof(link));

if(p1->ch < p2->ch)

{

headMerged->next = p1;

p1 = p1->next;

headMerged->next->next = NULL;

}

else

{

headMerged ->next= p2;

p2 = p2->next;

headMerged->next->next= NULL;

}

//后续节点比较

link \* pTemp = headMerged->next;

while( NULL!= p1&& NULL!= p2)

{

if(p1->ch < p2->ch)

{

pTemp->next = p1;

p1 = p1->next;

pTemp = pTemp->next;

pTemp->next = NULL;

}

else

{

pTemp->next = p2;

p2 = p2->next;

pTemp = pTemp->next;

pTemp->next = NULL;

}

}

//后续的接替

if(p1 != NULL)

pTemp->next = p1;

else if(p2 != NULL)

pTemp->next = p2;

return headMerged;

}

//上一道题的递归解法

link \* MergeSortedList\_by\_recursion(link \* head1, link \* head2)

{

//即使初始化也是边界条件

link \*p1= head1;

link \*p2= head2;

if(p1== NULL)

return head2;

if(p2 == NULL)

return head1;

link \* headMerged = NULL,\*p=NULL;

if(p1->ch < p2->ch)

{

p= p1;

p->next = MergeSortedList\_by\_recursion(p1->next, p2);

}

else

{

p= p2;

p->next = MergeSortedList\_by\_recursion(p1, p2->next);

}

return p;

}

int main()

{

link \*apH = NULL,\*aMed=NULL,\*bpH = NULL,\*MergepH=NULL,\*rever\_a=NULL;

char a[]="12345";

char b[]="25789";

apH=create\_link(a);

bpH=create\_link(b);

printf("original a link is: ");

print\_link(apH);

printf("a link has %d point\n",GetListLength(apH) ) ;

rever\_a=ReverseList\_by\_recursion(apH);

printf("Reverse a link is: ");

print\_link(rever\_a);

aMed=GetMiddleNode(apH);

printf("med point is: %c\n",aMed->ch);

ReverseList\_by\_recursion(apH);

print\_link(apH);

print\_link(bpH);

MergepH=MergeSortedList\_by\_recursion(apH,bpH);

printf("ab mergeed link is: ");

print\_link(MergepH);

return 0;

}