4 建立有序链表

void list\_sort\_insert(pstu\* pphead, pstu\* pptail, int i) {

pstu pcur, ppre, pnew;

pnew = (pstu)malloc(sizeof(stu));

memset(pnew, 0, sizeof(stu));

pnew->num = i;

pcur = \*pphead;

ppre = \*pphead;

if (NULL == pcur) {

\*pphead = pnew;

\*pptail = pnew;

}

else if (i < pcur->num) {

pnew->pNext = pcur;

\*pphead = pnew;

}

else {

while (pcur != NULL) {

if (pcur->num > i) {

ppre -> pNext = pnew;

pnew->pNext = pcur;

break;

}

ppre = pcur;

pcur = pcur->pNext;

}

if(NULL==pcur)

{

(\*pptail)->pNext = pnew;

\*pptail = pnew;

}

}

}

int main() {

pstu p;

pstu phead = NULL, ptail = NULL; // 代表链表；

int i;

float f;

printf("建立链表: \n");

while (scanf("%d", &i) != EOF) { // 链表的头插，尾插，有序插入

//list\_head\_insert(&phead, &ptail, i);

list\_sort\_insert(&phead, &ptail, i);

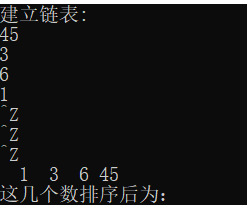
}

int k;

printf("这几个数排序后为： \n");

list\_print(phead);

}



5 删除指定的某个结点。

void list\_delete(pstu\* pphead, stu\*\* pptail, int delete\_num) {

pstu pcur, ppre;

pcur = \*pphead;

ppre = pcur;

if (pcur != NULL) {

if (pcur->num == delete\_num) {

\*pphead = pcur->pNext; // 改变头指针

if (NULL == \*pphead) { // 删除后链表为空

\*pptail = NULL;

}

free(pcur);

}else { // 删除中间结点

while (pcur!=NULL)

{

if (pcur->num == delete\_num) {

ppre->num = pcur->pNext;

free(pcur);

break;

}

ppre = pcur;

pcur = pcur->pNext;

}

if (NULL == ppre->pNext) {

\*pptail = ppre;

}

if (NULL == pcur) {

printf("no this node");

}

}

}

int main() {

pstu p;

pstu phead = NULL, ptail = NULL; // 代表链表；

int i;

float f;

printf("建立链表: \n");

while (scanf("%d", &i) != EOF) { // 链表的头插，尾插，有序插入

list\_head\_insert(&phead, &ptail, i);

}

int k;

list\_print(phead);

printf("请输入要删除的结点:\n");

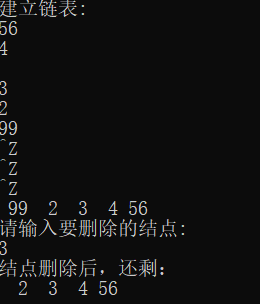
scanf("%d", &k);

list\_delete(&phead, &ptail, i);

printf("结点删除后，还剩： \n");

list\_print(phead);

}



6 将两个有序链表合并成一个有序链表

void list\_sort\_insert(pstu\* pphead, pstu\* pptail, int i) {

pstu pcur, ppre, pnew;

pnew = (pstu)malloc(sizeof(stu));

memset(pnew, 0, sizeof(stu));

pnew->num = i;

pcur = \*pphead;

ppre = \*pphead;

if (NULL == pcur) {

\*pphead = pnew;

\*pptail = pnew;

}

else if (i < pcur->num) {

pnew->pNext = pcur;

\*pphead = pnew;

}

else {

while (pcur != NULL) {

if (pcur->num > i) {

ppre->pNext = pnew;

pnew->pNext = pcur;

break;

}

ppre = pcur;

pcur = pcur->pNext;

}

if (NULL == pcur)

{

(\*pptail)->pNext = pnew;

\*pptail = pnew;

}

}

}

int main() {

pstu p;

pstu phead1 = NULL, ptail1 = NULL; // 代表链表；

pstu phead2 = NULL, ptail2 = NULL; // 代表链表；

int i,j;

float f;

printf("建立链表: \n");

while (scanf("%d", &i) != EOF) { // 链表的头插，尾插，有序插入

//list\_head\_insert(&phead, &ptail, i);

list\_sort\_insert(&phead1, &ptail1, i);

}

printf("第一个有序数列是： \n");

list\_print(phead1);

printf("-------------------------- \n");

printf("请着输入第二个有序数列： \n");

while (scanf("%d", &j) != EOF) { // 链表的头插，尾插，有序插入

//list\_head\_insert(&phead, &ptail, i);

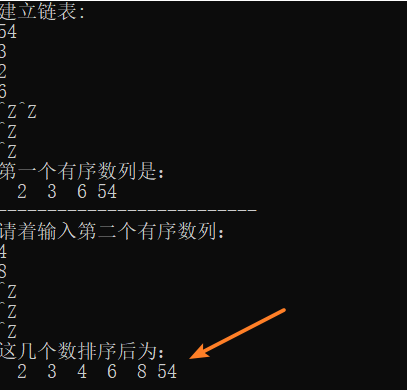
list\_sort\_insert(&phead1, &ptail1, j);

}

printf("这几个数排序后为： \n");

list\_print(phead1);

}



7 找出链表的倒数第四个节点

void list\_reverse(pstu\* pphead) {

p = \*pphead;

q = p->pNext;

r = p->pNext;

while (p != NULL)

{

r = p->pNext;

p->pNext = \*pphead;

\*pphead = p;

p = r;

}

}

int main() {

pstu p;

pstu phead1 = NULL, ptail1 = NULL; // 代表链表；

int i, j;

printf("建立链表: \n");

while (scanf("%d", &i) != EOF) { // 链表的头插，尾插，有序插入

//list\_head\_insert(&phead, &ptail, i);

list\_sort\_insert(&phead1, &ptail1, i);

}

printf("这几个数排序后为： \n");

list\_print(phead1);

// 将两个链表逆置

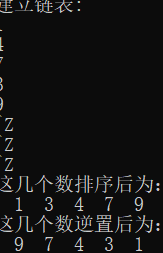
list\_reverse(&phead1);

//reverse\_list(&phead1);

printf("这几个数逆置后为： \n");

list\_print(phead1);

}



8 找出链表的倒数第四个节点

int main() {

int n1[7];

pstu p;

pstu phead1 = NULL, ptail1 = NULL; // 代表链表；

printf("建立链表: \n");

scanf("%d%d%d%d%d%d%d", &n1[0], &n1[1], &n1[2], &n1[3], &n1[4], &n1[5], &n1[6]);

for(int i=0;i<7;i++) { // 链表的头插

list\_head\_insert(&phead1, &ptail1,n1[i] );

}

pstu loPoint;

pstu hiPoint;

int cha=4;

loPoint = phead1;

hiPoint = phead1 ;

//list\_visit(, cha);

while (cha--) {

hiPoint = hiPoint->pNext;

}

while (hiPoint != NULL)

{

hiPoint = hiPoint->pNext;

loPoint = loPoint->pNext;

}

printf("链表倒数第四个结点值是： %d\n", loPoint->num);

}



9 找出链表的中间节点

int main() {

int n1[7];

pstu p;

pstu phead1 = NULL, ptail1 = NULL; // 代表链表；

printf("建立链表: \n");

scanf("%d%d%d%d%d%d%d", &n1[0], &n1[1], &n1[2], &n1[3], &n1[4], &n1[5], &n1[6]);

for (int i = 0; i < 7; i++) { // 链表的头插

list\_head\_insert(&phead1, &ptail1, n1[i]);

}

pstu loPoint;

pstu hiPoint;

loPoint = phead1;

hiPoint = phead1;

while (hiPoint->pNext != NULL)

{

hiPoint = hiPoint->pNext;

if (hiPoint != NULL) {

hiPoint = hiPoint->pNext;

}

loPoint = loPoint->pNext;

}

printf("链表中间结点是： %d\n", loPoint->num);

}



10 判断单链表是否有环

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include <time.h>

#include <stdbool.h>

typedef struct node {

int id;

struct node\* next;

}Node, \*LinkList;

void addNode(LinkList linkList, int id);

void initLinkList(Node\*\* linkList);

void randomNode(LinkList\* pNode, int i);

void randomCir(LinkList\* pNode, int i);

bool check\_cir(LinkList pNode);

bool check\_cir(LinkList pNode) { // 判断是否有环

Node\* pn1, \*pn2;

pn1 = pNode;

pn2 = pNode;

while (pn1->next && pn2->next)

{

pn1 = pn1->next;

if (pn2->next->next)

pn2 = pn2->next->next;

else

pn2 = pn2->next;

if (pn1 == pn2) {

printf("\n在环中相遇的链表结点是 :%d\n", pn1->id);

return true;

}

}

return false;

}

void randomCir(LinkList\* pNode, int i) { // 从产生的链表中随机随机产生环的入口

Node\* pn = (\*pNode)->next;

Node\* tail = (\*pNode)->next;

srand((unsigned)time(NULL));//置随机数种子

int k = rand() % i;//在 0-i内随机找一个位置

int n = 0;

i--;

while (i--)

{

tail = tail->next;

}

while (pn->next && n < k)

{

n++;

pn = pn->next;

}

printf("\n环入口是: %d\n", pn->id);

tail->next = pn->next;

}

void randomNode(LinkList\* pNode, int i)

{

srand((unsigned)time(NULL)); //置随机数种子

for (int k = 0; k < i; k++) {

addNode(\*pNode, rand() % i); // 传入结点与id值

}

}

//用头插法

void addNode(LinkList linkList, int id)

{

Node\* n = (Node\*)malloc(sizeof(Node));

n->id = id;

n->next = linkList->next;

linkList->next = n;

printf(" %d ",n->id);

}

//链表的初始化操作

void initLinkList(Node\*\* linkList)

{

(\*linkList) = (Node\*)malloc(sizeof(Node));

(\*linkList)->next = NULL;

}

int main(void)

{

LinkList linkList;

initLinkList(&linkList);

printf(" number is :\n");

randomNode(&linkList, 10); //随机产生一个单链表,10个结点

randomCir(&linkList, 10); //随机产生一个环

if (check\_cir(linkList))

{

printf("有环\n");

}

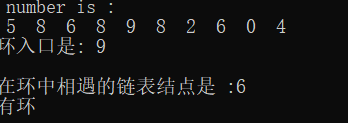
else {

printf("无环\n");

}

return 0;

}



11 判断两个链表是否相交， 如果相交， 计算交点

int linklength(LinkList phead)//求单链表长度

{

int k = 0;

while (phead != NULL)

{

k++;

phead = phead->next;

}

return k;

}

void print(LinkList linkList,int n)

{

int i;

for (i = 0; i < n; i++)

{

printf(" %3d ", linkList->id);

linkList = linkList->next;

}

printf("\n");

}

int main(void)

{

int len = 10;

LinkList linkList;

LinkList linkList2;

initLinkList(&linkList);

initLinkList(&linkList2);

randomNode(&linkList, len); //随机产生一个单链表,10个结点

//randomNode(&linkList2, len2);

linkList2->next = linkList->next->next->next;

int len1, len2;

len1 = linklength(linkList) - 1;

len2 = linklength(linkList2) - 1;

printf("len1 is %d\n len2 is %d\n", len1, len2);

printf("\n-------序列一---------\n");

print(linkList, len1);

printf("\n--------序列二--------\n");

for (int i = 0; i < (len1 - len2); i++) {

linkList = linkList->next;

}

print(linkList2, len2);

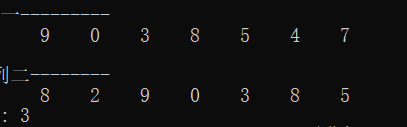
if (linkList->id == linkList2->id) {

printf("相交,交点是: %d", linkList2->next->next->next->id);

}

return 0;

}



12. 删除单链表中重复的元素

typedef struct node {

int val;

struct node\* next;

}Node, \*LinkList;

LinkList DeleteRepeateElement(Node\* L)

{

/\*

参数:链表的首地址

函数功能:在递增有序的单链表中删除重复节点

\*/

LinkList s;//指向待删除的节点

while (L->next != NULL)

{

if (L->val == L->next->val)//如果下一个节点与当前节点相当,删除下一个节点

{

s = L->next; // 令L下一个结点为s

L->next = L->next->next; //

free(s);

}

L = L->next;

}

return;

}

void list\_tail\_insert(LinkList\* pphead, LinkList\* pptail, int i) {

LinkList pnew;

pnew = (LinkList)malloc(sizeof(Node));

memset(pnew, 0, sizeof(Node));

pnew->val = i;

if (NULL == \*pptail) {

\*pphead = pnew;

\*pptail = pnew;

}

else {

(\*pptail)->next = pnew;

\*pptail = pnew;

}

}

void print(LinkList linkList)

{

int i;

while (linkList != NULL)

{

printf(" %3d ", linkList->val);

linkList = linkList->next;

}

printf("\n");

}

void main() {

int i,n=0;

LinkList phead = NULL, ptail = NULL;

while (scanf("%d", &i) != EOF) {

list\_tail\_insert(&phead, &ptail, i);

n++;

}

printf("输入时的序列为: ");

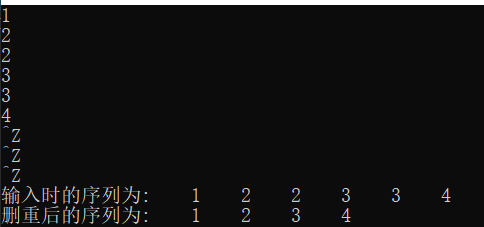
print(phead);

DeleteRepeateElement(phead);

printf("删重后的序列为: ");

print(phead);

}



13将一个链表拆分（将链表奇数位置上的节点构成一个链表，偶数位置上的节点构成另一

个链表）

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include<stdio.h>

#include<stdlib.h>

typedef struct linklist {

int data;

struct linklist\* next;

}list, \* plist;

void list\_tail\_sort(plist\* pphead, plist\* pptail, int key)

{

plist pnew = (plist)calloc(1, sizeof(list));

pnew->data = key;

if (NULL == \*pphead)

{

\*pphead = pnew;

\*pptail = pnew;

}

else {

(\*pptail)->next = pnew;

\*pptail = pnew;

}

}

void splitlist(plist\* pphead, plist\* pphead1, plist\* pphead2)

{

int count = 0;

plist ptail1=NULL, ptail2 = NULL;

plist pnew;

while (\*pphead)

{

count++;

pnew = \*pphead;

if (0==count % 2)

{

if (NULL == \*pphead1){ // 再用尾插法构插入新链表1

\*pphead1 = pnew;

ptail1 = pnew;

}

else {

ptail1->next = pnew;

ptail1 = pnew;;

}

}

else { // 再用尾插法构插入新链表2

if (NULL == \*pphead2){

\*pphead2 = pnew;

ptail2 = pnew;

}

else {

ptail2->next = pnew;

ptail2 = pnew;;

}

}

\*pphead = (\*pphead)->next; //指针后移

}

ptail1->next = NULL; // 将两个新链表尾结点置空

ptail2->next = NULL;

}

void list\_print(plist p)

{

while (p)

{

printf("%d ", p->data);

p = p->next;

}

printf("\n");

}

int main()

{

plist phead = NULL, ptail = NULL;

int key;

plist phead1 = NULL, phead2 = NULL;

while (scanf("%d", &key) != EOF)

{

list\_tail\_sort(&phead, &ptail, key);

}

printf("List：");

list\_print(phead);

splitlist(&phead, &phead1, &phead2);

printf("分离后的两个链表是：");

list\_print(phead);

printf("List1：");

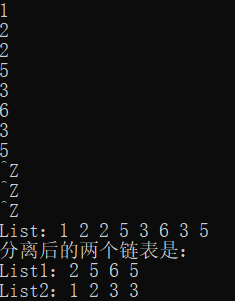
list\_print(phead1);

printf("List2：");

list\_print(phead2);

system("pause");

}



Day7

1 完成栈的压栈，弹栈，获取栈顶元素，判断栈是否为空，和获取栈大小

void init\_stack(pStack s)

{

memset(s,0,sizeof(Stack));

}

void push(pStack s,ElemType val)

{

pNode pnew=(pNode)calloc(1,sizeof(Node));

pnew->m\_val=val;

if(!s->phead) //栈为空，没有栈顶元素

{

s->phead=pnew; //栈头更新

}else{

pnew->next=s->phead; // 将新结点 指 向栈顶元素的 值 （pnew覆盖原栈顶元素）

s->phead=pnew; //栈头指针更新

}

s->size++; // 栈大小更新

}

ElemType top(pStack s)

{

if (!s->size)

{

printf("stack is empty\n");

return;

}

return s->phead->m\_val;

}

ElemType pop(pStack s)

{

//判断栈是否为空

pNode pcur=s->phead; // temp结点

if(!s->size)

{

printf("stack is empty\n");

return;

}

s->phead=pcur->next; //栈头指针更新

s->size--;

return pcur->m\_val;

//free(pcur); //因为取的指针（引用），所以不用再return

}

int empty(pStack s)

{

return !s->size; //

}

int stack\_size(pStack s)

{

return s->size; // size是栈结构体的值

}

void use\_stack()

{

Stack s;

init\_stack(&s); // 对栈初始化

push(&s,2);

push(&s,3);

push(&s, 5);

printf("stack size=%d\n",stack\_size(&s)); // 栈大小

printf("top is: %d\n", top(&s));

printf("pop:%d\n", pop(&s));

printf("top is: %d\n",top(&s));

printf("%d（0 is not empty）\n",empty(&s));

}

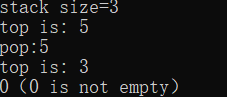
int main()

{

use\_stack();

system("pause");

}



2 二叉树层次建树并先序输出

typedef struct node{

char c;

struct node \*pleft;

struct node \*pright;

}TNode,\*pTNode;

typedef struct queue\_t{

pTNode insert\_pos;

struct queue\_t \*pNext;

}Queue\_t,\*pQueue\_t;

#define N 10

void main()

{

pTNode tree\_root = NULL;

char c[N + 1] = "ABCDEFGHIJ";

pTNode p[N + 1];

int i, j; //记住往哪一个节点放数据

for (i = 0; i < N; i++)

{

p[i] = (pTNode)calloc(1, sizeof(TNode)); //p[i]存放的新结点的地址

p[i]->c = c[i]; //p[i]的值为c[i]

}

for (i = 0; i < N; i++)

{

if (NULL == tree\_root)//如果树为空

{

tree\_root = p[i]; // tree\_root 就有值了

j = 0; //

}

else {

if (NULL == p[j]->pleft)//判断要插入的节点左边是否可以放节点

{

p[j]->pleft = p[i];

}

else if (NULL == p[j]->pright)

{

p[j]->pright = p[i];

j++;

}

}

}

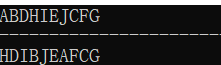
pre\_order(tree\_root);

printf("\n-------------------------\n");

mid\_order(tree\_root);

printf("\n-------------------------\n");

}



3.1 选择排序

#define N 5

int main()

{

int arr[N] = { 0,13,45,7,4 };

printf("输入是： ");

for (int i = 0; i < N; i++) {

printf(" %d ", arr[i]);

}

printf("\n");

int i, j, min\_pos;

for (i = 0; i < N - 1; i++)

{

min\_pos = i; // 相当于temp，记录最小位置，每轮外层循环更新

for (j = i + 1; j < N; j++)

{

if (arr[min\_pos] > arr[j])

{

min\_pos = j; // 位置交换

}

}

{ // 交换

int tmp;

tmp = arr[i];

arr[i] = arr[min\_pos];

arr[min\_pos] = tmp;

}

}

printf("选择插入排序后： ");

for (int i = 0; i < N; i++) {

printf(" %d ", arr[i]);

}

system("pause");

}



3.2 插入排序

#define N 5

int main()

{

int arr[N] = { 0,13,45,7,4 };

//arr\_insert(arr);

int i, j, insert\_val;

for (i = 1; i < N; i++)//外层控制要插入的数据

{

insert\_val = arr[i];//把要插入的元素存起来

for (j = i - 1; j >= 0; j--) // 倒着比较

{

if (insert\_val < arr[j]) // 如果满足 元素后移

{

arr[j + 1] = arr[j];

}

else {

break; //否则 不用管

}

}

arr[j + 1] = insert\_val; // 再最后空隙中插入这个数

}

for (int i = 0; i < N; i++) {

printf(" %d ", arr[i]);

}

system("pause");

}

