////第一课 课程导学，编程环境搭建

////第二课 数据的类型，输入/输出

//#include <stdio.h>

//

//int main()

//{

//// //数据类型、混合运算、进制转换

//// printf("%c, %d, %c, \n", 'A'+32, 'A', '\0');

////

//// int n1 = 5;

//// float n2 = 5/2;

//// float n3 = (float)5/2;

//// printf("%f %f\n", n2, n3);

////

//// int n = 0x7b; //=0173 =123

//// printf("%d, %o, %x, \n", n, n, n);

//

//

// //标准输入

// int i = 0;

// char c;

// float f = 0;

//

// scanf("%d", &i);

// printf("i = %c\n", i);

//

// fflush(stdin); //清空标准输入缓冲区

// scanf("%c", &c);

// printf("c = %d\n", c);

////

//// scanf("%f", &f);

//// printf("f = %f\n", f);

////

//// scanf("%f", &f);

//// printf("f = %f\n", f);

//// //只留下一个'\n'

//// getchar(); //getchar();只能清除一个字符,上一个输入不包含空格时可用

//// scanf("%c", &c);

//// printf("c = %c\n", c);

//

//// int i, ret;

//// float f;

//// char c;

////

////// ret = scanf("%d%c%f", &i, &c, &f); //输入使用空格隔开 空格=‘32’

////// printf("i=%d,c=%c,f=%f,ret=%d\n", i, c, f, ret); //ret=2,匹配成功2个

////

//// ret = scanf("%d %c%f", &i, &c, &f); //空格抵消

//// printf("i=%d,c=%c,f=%f,ret=%d\n", i, c, f, ret); //ret=3,匹配成功3个

//

//

//

// return 0;

//}

////第三课 运算符和表达式

//#include <stdio.h>

//

//

//int main()

//{

// //arithmetic 算术运算符

// printf("%d\n", 10/2%4);

//

//// //relation 关系运算符

//// int a = 0;

//// while(scanf("%d", &a))

//// {

//// if(3<a<10) //a>3 && a<10

//// {

//// printf("a is between 3 and 10\n");

//// }

//// else

//// {

//// printf("a is not between 3 and 10\n");

//// }

//// }

//

//// //logic 逻辑运算符—— 判断闰年

//// int y=0;

//// while(scanf("%d", &y))

//// {

//// if(y%400==0 || y%4==0 && y%100!=0)

//// {

//// printf("is a leap year\n");

//// }else{

//// printf("is not a leap year\n");

//// }

//// }

//// int i=6;

//// printf("%d\n", !!i);

//

// //short circuit 短路运算

// int j=1;

// j&&printf("you can't see me !\n");

//

// //assign 赋值操作符

// int a=1,b=2;

// printf("%d %d\n", a+=2,b\*=5);

//

// //求字节长度运算符 sizeof

// printf("%d %d\n", sizeof(char), sizeof(a));

//

// return 0;

//}

////第四课 选择/循环语句

//#include <stdio.h>

//

//int main(){

//

//// //条件/选择/分支语句 if-else

//// int i;

//// while(scanf("%d", &i))

//// {

//// if(i>0)

//// {

//// printf("i is bigger than 0\n");

//// }else{

//// printf("i is not bigger than 0\n");

//// }

//// }

//

// //循环语句 while for continue break

// int i=1,total=0;

// while(i<=100)

// {

// total+=i;

// i++;

// }

// printf("%d\n", total);

//

// return 0;

//}

////第五课 一维数组和字符数组

//#include <stdio.h>

//

//void print(int a[], int length);

//void printc(char a[]);

//int main(){

//// //访问越界

//// int a[5]={1,2,3,4,5};

//// int j=20;

//// int i=10;

////

//// a[5]=6;

//// a[6]=7;

//// printf("%d\n", i);//i未被赋值、值被改变了

//

// //数组传递

// int a[5]={1,2,3,4,5};

// int l=sizeof(a)/sizeof(0);

// print(a, l);

// scanf("%d", &l);

// printf("%d\n", l);

//

// //数组打印

// char arr1[6]="hello";//结束符占一空间‘\0’

// char arr2[5]={'h','e','l','l','o'};//越界

// printc(arr1);

// printc(arr2);

//

// //scanf读取字符串，自动补0，空格结束输入，不读取空格

// char arr3[10]={0};

// scanf("%s", arr3);

// printf("%s\n", arr3);

//

// //strlen();strcpy();strcmp();strcat();

//

// return 0;

//}

//

////数组传递

//void print(int a[], int length)

//{

// int i=0;

// //数组传递后无法计算数组大小sizeof(a)/sizeof(0)

// for(i=0; i<length; i++)

// {

// printf("%d\n", a[i]);

// }

// a[3]=10;

// printf("a[3]=%d\n", a[3]);

//

//

//}

//

////模拟printf("%s", arr);

//void printc(char a[])

//{

// int i=0;

// for(i=0; a[i]!='\0'; i++)

// {

// printf("%c", a[i]);

// }

// printf("\n");

//}

////第六课 指针

//#include <stdio.h>

//

//void change1(int i);

//void change2(int \*j);

//

//int main(){

// //间接访问

// int i=0;

// printf("%d\n", i);

// change1(i); //C语言的调用是值传递，形参赋值给形参，j=i

// printf("%d\n", i);

// change2(&i); //指针应用场景-间接访问

// printf("%d\n", i);

//

// return 0;

//}

//

//void change1(int j)

//{

// j=2;

//}

//void change2(int \*k)

//{

// \*k=3;

//}

//————————————————————————————————————————————————

////指针应用场景-偏移

//#define N 5

//int main(){

// int a[N]={1,3,5,7,9};

// int \*pa=a;

// int i;

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

// {

// printf("%d ", \*(pa+i)); //+优先级低于\*

// }

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

//

// pa=&a[4]; //指向数组最后一个元素，

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

// {

// printf("%d ", \*(pa-i)); //+优先级低于\*

// }

//

//

// return 0;

//}

//————————————————————————————————————————————————

////指针与一维数组的传递

//void change(char \*p){

// \*p='H';

// \*(p+1)='E';

// p[2]='L';

//}

//

//int main(){

// char c[10]="hello";

// change(c);

// puts(c);

//

// return 0;

//}

//——————————————————————————————————————————————————

////指针与动态内存申请-malloc

//#include <stdlib.h>

//#include <string.h>

//

//char\* print\_stack()

//{

// char c[100]="I am print\_stack function";

// //puts(c);

// return c;

//}

//

//char\* print\_heap()

//{

// char \*c=(char\*)malloc(100);

// strcpy(c, "I am print\_heap function");

// return c;

//}

//

//int main(){

// //内存空间的动态申请和释放

// int size=20;

// char \*p;

// //scanf("%d", &size);//输入要申请的空间大小

// //malloc 返回 void\* 无类型指针

// p=(char\*)malloc(size);

// strcpy(p, "malloc success");

// puts(p);

// free(p); //释放申请的空间

// printf("free success\n");

// puts("free success");

//

// //栈空间的错误使用

// char \*pr;

// pr=print\_stack();

// puts(pr); //此处输出为NULL，因为函数申请的栈空间已被释放

// pr=print\_heap();

// puts(pr);

// free(pr);

//

// return 0;

//}

////第七课 函数

//#include <stdio.h>

//

////求n的阶层

//int fac(int n)

//{

// if(n>1){

// return n\*fac(n-1);

// }

// return 1;

//}

//

////青蛙跳台阶问题求解，类似斐波那契数列

//int flog(int n)

//{

// if(1==n || 2==n)

// {

// return n;

// }

// return flog(n-1)+flog(n-2);

//

//}

//

//int main(){

// int n=5;

// printf("%d\n", fac(n));

// printf("%d\n", flog(n));

//

// return 0;

//}

////第八课 结构体和C++引用

//#include <stdio.h>

//

//struct student{

// int num;

// char name[20];

// char sex;

// float score;

//}; //结构体类型声明，一般在函数外，工程中常放在头文件中

//

////typrdef

//typedef struct student2{

// int num;

// char name[20];

//}stu2, \*pstu2;

//

////通过引用直接修改主函数参数，或一级指针变量

//void modify\_pointer(int \*&p, int \*&q){

// p=q;

//}

//

//int main(){

// //结构体初始化，非初始化则只能单独赋值

// struct student s={1001, "lele", 'M', 89.9};

// //结构体数组赋值

// struct student sarr[3]={1001,"lilei",'M',90.5, 1005,"zhangsan",'M',88.5, 1007,"lili",'F',91};

// //结构体的调用

// int i=0;

// for(i=0; i<3; i++)

// {

// //scanf("%d%s %c%f", &sarr[i].num, sarr[i].name, &sarr[i].sex, &sarr[i].score);

// }

// for(i=0; i<3; i++)

// {

// printf("%d %s %c %f\n", sarr[i].num, sarr[i].name, sarr[i].sex, sarr[i].score);

// }

// printf("\n");

//

// //结构体指针

// struct student \*ps=&s;//struct student \*ps

// printf("%d %s %c %f\n", (\*ps).num, (\*ps).name, (\*ps).sex, (\*ps).score);

// printf("%d %s %c %f\n", ps->num, ps->name, ps->sex, ps->score);

// ps=sarr;

// printf("%d %s %c %f\n", ps->num, ps->name, ps->sex, ps->score);

// ps=ps+1;

// printf("%d %s %c %f\n", ps->num, ps->name, ps->sex, ps->score);

// printf("\n");

//

// //typedef

// stu2 s2={1010, "wangwei"};

// printf("%d %s\n", s2.num, s2.name);

// pstu2 ps2=&s2;

// printf("%d %s\n", ps2->num, ps2->name);

// printf("\n");

//

// //C++引用 &

// int \*p=NULL; //NULL不允许访问

// int n=10;

// int \*q=&n;

// modify\_pointer(p,q);

// printf("\*q=%d after \*p=%d\n", \*q, \*p);

//

// return 0;

//}

////第九课 数据结构概述

//#include <stdio.h>

//

////时间复杂度为 O(nlog2n）

//int main(){

// int i,x=2;

// int n=100;//问题规模

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

// {

// x=1;

// while(x<n/2)

// {

// x\*=2;

// }

// }

//

// return 0;

//}

////第十课 线性表代码实战

//#include <stdio.h>

//

////--顺序表--

//#define MaxSize 50 //顺序表的最大空间

//typedef int ElemType; //快速修改代码存储其他类型变量

////顺序表结构体类型声明，静态分配存储空间

//typedef struct{

// ElemType data[MaxSize];

// int length;

//}SqList;

//

////顺序表打印，只能打印%d

//void ListPrint(SqList L){

// for(int i=0; i<L.length; i++)

// {

// printf("%3d", L.data[i]);

// }

// printf("\n");

//}

//

////顺序表插入一个元素，使用引用

//bool ListInsert(SqList &L, int pose, ElemType inse){

// //判断pos是否合法

// if(pose<1 || pose>L.length+1)

// {

// return false;

// }

// //判断存储空间是否已满

// if(L.length==MaxSize)

// {

// return false;

// }

// for(int i=L.length; i>=pose; i--)

// {

// L.data[i]=L.data[i-1];

// }

// L.data[pose-1]=inse;

// L.length++;

// return true;

//}

//

////顺序表删除一个元素

//bool ListDelete(SqList &L, int pose, ElemType &dele){

// //判断pos是否合法

// if(pose<1 || pose>L.length)

// {

// return false;

// }

// dele=L.data[pose-1];

// for(int i=pose; i<L.length; i++)

// {

// L.data[i-1]=L.data[i];

// }

// L.length--;

// return true;

//}

//

////顺序表查找一个元素，按值查找(顺序表中元素各不相同)

//int ListLocate(SqList L, ElemType elem){

// for(int i=0; i<L.length; i++)

// {

// if(elem==L.data[i])

// {

// return i+1;

// }

// }

// return 0;

//}

//

//

//int main(){

//

// SqList L; //定义一个顺序表

// L.data[0]=1; //初始化

// L.data[1]=2;

// L.data[2]=3;

// L.length=3;

// ListPrint(L);

//

// bool ret=0; //存放返回值

// //插入一个元素

// ret=ListInsert(L, 2, 60);

// if(ret)

// {

// printf("sqlist insert success\n");

// ListPrint(L);

// }

// else{

// printf("sqlist insert failed\n");

// }

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

//

// //删除一个元素

// ElemType del=0; //存放删除的元素

// ret=ListDelete(L, 3, del);

// if(ret)

// {

// printf("sqlist delete success, del=%d\n", del);

// ListPrint(L);

// }

// else{

// printf("sqlist delete failed\n");

// }

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

//

// //按值查找一个元素

// int pos=0;

// pos=ListLocate(L, 60);

// if(pos)

// {

// printf("sqlist locate success, pos=%d\n", pos);

// ListPrint(L);

// }

// else{

// printf("sqlist locate failed\n");

// }

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

//

//

// return 0;

//}

////第十一课 链表实战

//#include <stdio.h>

//#include <stdlib.h>

//

////--链表1-新建、查找、插入、删除--

//typedef int ElemType;

////链表结构体类型声明

//typedef struct LNode{

// ElemType data;

// struct LNode \*next;

//}LNode, \*LinkList;

//

////头插法建立链表，输入1 3 5 999，读到999时停止

//void list\_head\_insert(LinkList &L) {//LinkList=LNode\*

// L=(LinkList)malloc(sizeof(LNode));

// L->next=NULL;

// LinkList l;

// ElemType input;

//

// scanf("%d", &input); //只能输入int类型，不能根据ElemType改变

// while(input!=999)

// {

// l=(LinkList)malloc(sizeof(LNode));

// l->data=input;

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

// L->next=l;

// scanf("%d", &input);

// }

//

//}

//

////打印链表

//void list\_print(LinkList L){

// L=L->next;

// while(L)

// {

// printf("%3d", L->data);

// L=L->next;

// }

// printf("\n");

//}

//

////尾插法建立链表

//void list\_tail\_insert(LinkList &L){

// L=(LinkList)malloc(sizeof(LNode));

// LinkList l,end=L;

// ElemType input;

//

// scanf("%d", &input);

// while(input!=999)

// {

// l=(LinkList)malloc(sizeof(LNode));

// l->data=input;

// end->next=l;

// end=l;

// scanf("%d", &input);

// }

// end->next=NULL;

//}

//

////按位查找

//LinkList ElemGet(LinkList L, int n){

// int i=0;

// if(n<0)

// {

// return NULL;

// }

// while(i<n && L)

// {

// L=L->next;

// i++;

// }

// if(L)

// {

// return L;

// }else{

// return NULL;

// }

//

//}

//

////按值查找,只能查得遇上的第一个

//LinkList ElemLocate(LinkList L, ElemType data){

// while(L)

// {

// if(data==L->data)

// {

// break;

// }

// L=L->next;

// }

// return L;

//}

//

////第i个位置插入元素i>0；建立链表时已为头结点申请空间，此处不引用

//bool ElemInsert(LinkList L, int i, ElemType put){

// LinkList p=ElemGet(L, i-1);

// if(p){

// LinkList l;

// l=(LinkList)malloc(sizeof(LNode));

// l->data=put;

// l->next=p->next;

// p->next=l;

// return true;

// }else{

// return false;

// }

//

//}

//

////删除第i个元素，并取出；此时L不改变，不用引用

//bool ElemDelete(LinkList L, int i, ElemType &get){

// LinkList p=ElemGet(L, i-1);

// if(p)

// {

// LinkList l=p->next;

// get=l->data;

// p->next=l->next;

// free(l);

// l=NULL; //避免野指针

// return true;

// }else{

// return false;

// }

//}

//

//

//int main(){

// LinkList L; //链表头

//

// //链表的建立

//// list\_head\_insert(L);

//// list\_print(L);

// list\_tail\_insert(L);

// list\_print(L);

//

// //元素按位查找

// LinkList n=NULL;

// n=ElemGet(L, 4);

// if(n){

// printf("%3d\n", n->data);

// }else{

// printf("NULL\n");

// }

// //元素按值查找

// n=ElemLocate(L, 3);

// if(n){

// printf("%3d\n", n->data);

// }else{

// printf("NULL\n");

// }

// //元素插入

// bool ret;

// list\_print(L);

// ret=ElemInsert(L, 3, 66);

// if(ret){

// list\_print(L);

// }else{

// printf("insert failed\n");

// }

// //元素删除

// ElemType del=0;

// ret=ElemDelete(L, 4, del);

// if(ret){

// list\_print(L);

// }else{

// printf("delete failed\n");

// }

// return 0;

//}

//

////第十二课 链表

//#include <stdio.h>

//#include <stdlib.h>

//

////--链表2-真题实战--

////链表采用带头结点的单链表保存，重新排列L中的各结点

////设计一个空间复杂度为O(1)，时间上尽可能高效的算法

//

////基本思想：

////1.找到中间节点

////2.后半链表L2原地逆置

////3.合并L和L2

//typedef int ElemType;

//typedef struct LNode{

// ElemType data;

// struct LNode \*next;

//}LNode,\*LinkList;

//

//

////尾插法新建链表

//void list\_tail\_insert(LinkList &L){

// L=(LinkList)malloc(sizeof(LNode));

// LinkList l, end=L;

// ElemType input;

//

// scanf("%d", &input);

// while(input!=999)

// {

// l=(LinkList)malloc(sizeof(LNode));

// l->data=input;

// end->next=l;

// end=l;

// scanf("%d", &input);

// }

// end->next=NULL;

//

//}

//

////打印链表

//void list\_print(LinkList L){

// L=L->next;

// if(NULL==L){

// printf("%3c", '\*');

// } //若链表为空，打印\*

//

// while(L){

// printf("%3d", L->data);

// L=L->next;

// }

//

// printf("\n");

//}

//

////双指针法，前半长度大于等于后半,一个节点时L2->next=NULL,O(n/2)

//void list\_find\_middle(LinkList L, LinkList &L2){

// L2=(LinkList)malloc(sizeof(LNode));

// LinkList p1, p2;

// p1=p2=L->next;

// if(NULL==p2) //若链表为空，L2指向NULL，而不是L->next->next

// {

// L2->next=NULL;

// return;

// }

//

// while(p2)

// {

// p2=p2->next;

// if(NULL==p2)

// {

// break;

// }

// p2=p2->next;

// if(NULL==p2)

// {

// break;

// }

// p1=p1->next;

// }

//// L2=p1; //错误示范

// L2->next=p1->next;

// p1->next=NULL;

//

//}

//

////原地逆置L2,O(n/2)

//void list\_reverse(LinkList L2){

// LinkList p1, p2, p3;

// p1=L2->next;

// if(NULL==p1)

// {

// return;

// }

// p2=p1->next;

// if(NULL==p2)

// {

// return;

// }

// p3=p2->next;

//

// while(p3){

// p2->next=p1;

// p1=p2;

// p2=p3;

// p3=p3->next;

// }

//

// p2->next=p1;

// L2->next->next=p3; //NULL

// L2->next=p2;

//// p1->next=p3; //问题所在 NULL

//

//}

//

////轮流合并,O(n/2)

//void list\_merge(LinkList L, LinkList L2){

// LinkList pcur, p1, p2; //p1遍历L，p2遍历L2

// pcur=L->next;

// p2=L2->next;

// p1=pcur->next;

//

// while(p1 && p2){

// pcur->next=p2;

// pcur=p2; //pcur=pcur->next

// p2=p2->next;

// pcur->next=p1;

// pcur=p1; //pcur=pcur->next

// p1=p1->next;

// }

//

// if(p2){

// pcur->next=p2;

//// p2->next=NULL; //可以忽略

// }

//// else{

//// pcur->next=NULL; //可以忽略

//// }

//// if(p1){

//// pcur->next=p1; //因为只可能L2剩1结点所以可以忽略

//// p1->next=NULL; //可以忽略

//// }

//

//}

//

////算法时间复杂度为O(n)

//int main(){

// LinkList L;

// list\_tail\_insert(L);

// list\_print(L);

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

// //寻找中间结点，并返回

// LinkList L2;

// list\_find\_middle(L, L2);

// list\_print(L);

// list\_print(L2);

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

// //逆置链表

// list\_reverse(L2);

// list\_print(L2);

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

// //合并链表

// list\_merge(L, L2);

// free(L2);

// list\_print(L);

//

// return 0;

//}

////第十三课 栈与队列 顺序栈初始化、判断是否为空、压栈、获取栈顶元素、弹栈

//#include <stdio.h>

//#include <stdlib.h>

//

//#define SqsMaxSize 10

//typedef int SqsElemType;

//typedef struct SqStack{

// SqsElemType data[SqsMaxSize];

// int top;

//}SqStack;

////初始化栈

//void stack\_init(SqStack &S){

// S.top=-1;

//}

////判断栈空,空返回1

//bool stack\_isempty(SqStack S){

// if(-1==S.top){

// return true;

// }

// return false;

//}

////压栈

//bool stack\_push(SqStack &S, SqsElemType in){

// if(SqsMaxSize-1==S.top){

// return false;

// }

// S.data[++S.top]=in;

// return true;

//}

////获取栈顶元素

//bool stack\_gettop(SqStack S, SqsElemType &out){

// if(stack\_isempty(S)){

// return false;

// }

// out=S.data[S.top];

// return true;

//}

////弹栈

//bool stack\_pop(SqStack &S, SqsElemType &out){

// if(stack\_isempty(S)){

// return false;

// }

// out=S.data[S.top--];

// return true;

//}

//

//int main(){

// SqStack S;

// stack\_init(S);

// bool flag;

// //判断栈空

// flag=stack\_isempty(S);

// if(flag){

// printf("is not empty\n");

// }else{

// printf("is empty\n");

// }

// //入栈

// stack\_push(S, 3);

// stack\_push(S, 4);

// stack\_push(S, 5);

// //获取栈顶元素

// SqsElemType TopElem;

// flag=stack\_gettop(S, TopElem);

// if(flag){

// printf("%d\n", TopElem);

// }else{

// printf("is full\n");

// }

// //出栈

// SqsElemType PopElem;

// flag=stack\_pop(S, PopElem);

// if(flag){

// printf("%d\n", PopElem);

// }else{

// printf("is empty\n");

// }

// //入栈

// flag=stack\_gettop(S, TopElem);

// if(flag){

// printf("%d\n", TopElem);

// }else{

// printf("is full\n");

// }

//

// return 0;

//}

////第十三课 栈与队列 队列

//#include <stdio.h>

//#include <stdlib.h>

//

////循环顺序队列-初始化、是否空、入队、出队

//#define SqqMaxSize 5

//typedef int SqqElemType;

//typedef struct SqQueue{

// SqqElemType data[SqqMaxSize]; //存储 MaxSize-1 个元素

// int front, rear;

//}SqQueue;

////初始化

//void sqqueue\_init(SqQueue Q){

// Q.front=Q.rear=0;

//}

////判断是否为空,为空返回1

//bool sqqueue\_isempty(SqQueue Q){

// return Q.front==Q.rear;

//}

////入队

//bool sqqueue\_en(SqQueue &Q, SqqElemType in){

// if((Q.rear+1)%SqqMaxSize==Q.front){

// return false;

// }

// Q.data[Q.rear]=in;

// Q.rear=(Q.rear+1)%SqqMaxSize;

// return true;

//}

////出队

//bool sqqueue\_de(SqQueue &Q, SqqElemType &out){

// if(sqqueue\_isempty(Q)){

// return false;

// }

// out=Q.data[Q.front];

// Q.front=(Q.front+1)%SqqMaxSize;

// return true;

//}

//

////链队列

//typedef int LiqElemType;

//typedef struct LinkNode{

// LiqElemType data;

// struct LinkNode \*next;

//}LinkNode;

//typedef struct{

// LinkNode \*front, \*rear;

//}LinkQueue;

////链队初始化 带头结点的链表实现

//void linkqueue\_init(LinkQueue &LQ){

// LQ.front=LQ.rear=(LinkNode\*)malloc(sizeof(LinkNode));

//// LQ.front=LQ.rear=(LinkNode\*)malloc(sizeof(LinkNode));

//// LQ.rear=(LinkNode\*)malloc(sizeof(LinkNode));

// LQ.front->next=NULL;

//// LQ.rear=LQ.front;

//}

////队列为空返回1

//bool linkqueue\_isempty(LinkQueue LQ){

// if(LQ.rear==LQ.front){

// return true;

// }

// return false;

//}

////入队

//void linkqueue\_en(LinkQueue &LQ, LiqElemType in){

//// if(0){

//// return false;

//// }

// LinkNode \*lq=(LinkNode\*)malloc(sizeof(LinkNode));

// lq->data=in;

// lq->next=NULL; //链队遍历结尾

// LQ.rear->next=lq;

// LQ.rear=lq;

//// return true;

//}

////出队

//bool linkqueue\_de(LinkQueue &LQ, LiqElemType &out){

// if(linkqueue\_isempty((LQ))){

// return false;

// }

// LinkNode \*lq=LQ.front->next;

// out=lq->data;

// LQ.front->next=lq->next; //lq->next的作用

// if(LQ.rear==lq){

// LQ.rear=LQ.front;

// }//链表只剩一个结点时，被删除前，需要改变rear

// free(lq);

// return true;

//}

//

//

//

//int main(){

// //循环队列

// SqQueue Q;

// sqqueue\_init(Q);

// bool ret;

//

// //入队

// sqqueue\_en(Q, 3);

// sqqueue\_en(Q, 4);

// sqqueue\_en(Q, 5);

// ret=sqqueue\_en(Q, 6);

// if(ret){

// printf("%d\n", Q.data[Q.rear-1]);

// }else{

// printf("is full\n");

// }

// ret=sqqueue\_en(Q, 7);

// if(ret){

// printf("%d\n", Q.data[Q.rear-1]);

// }else{

// printf("is full\n");

// }

// //出队

// SqqElemType out;

// ret=sqqueue\_de(Q, out);

// if(ret){

// printf("%d\n", out);

// }else{

// printf("is empty\n");

// }

// ret=sqqueue\_en(Q, 8);

// if(ret){

// printf("%d\n", Q.data[Q.rear-1+SqqMaxSize]);

// }else{

// printf("is full\n");

// }

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

//

// //链队列

// LinkQueue LQ;

// linkqueue\_init(LQ);

// linkqueue\_en(LQ, 3);

// linkqueue\_en(LQ, 4);

// linkqueue\_en(LQ, 5);

// linkqueue\_en(LQ, 6);

// linkqueue\_en(LQ, 7);

// printf("%d\n", LQ.front->next->data);

// printf("%d\n", LQ.rear->data);

// LiqElemType element;

// ret=linkqueue\_de(LQ, element);

// if(ret){

// printf("%d\n", element);

// }else{

// printf("is empty\n");

// }

//

//

// return 0;

//}

////第十三课 栈与队列 队列

//#include <stdio.h>

//#include <stdlib.h>

//

////循环链式队列

////满足1.初始时为空，2.入队时允许增加占空空间，

////3.出队后空间可重复使用，占用空间只增不减，4.操作时间复杂度为O(1)

//

////链表结点定义

//typedef int ElemType;

//typedef struct LinkNode{

// ElemType data;

// struct LinkNode \*next;

//}LinkNode, \*LinkList;

////循环链队入队

//void EnQueue(LinkList front, LinkList &rear, ElemType val){

// LinkList pnew;

// //队满则申请空间，front放数据、rear空着

// if(rear->next==front){

// pnew=(LinkList)malloc(sizeof(LinkNode));

// rear->data=val;

// rear->next=pnew;

// rear=pnew;

// rear->next=front; //循环链队与链队的区别

// }else{

// //队不满则直接放入

// rear->data=val;

// rear=rear->next;

// }

//}

////循环链队出队

//void DeQueue(LinkList &front, LinkList rear){

// if(front==rear){

// printf("Linkqueue is empty\n");

// }else{

// printf("%d\n", front->data);

// front=front->next;

// }

//}

////循环链队操作总流程

//void CircleQueue(LinkList &front, LinkList &rear){

// //队头队尾指向同一个结点时，队列既是空的也是满的

// rear=front=(LinkList)malloc(sizeof(LinkNode));

// rear->next=front; //循环队列特殊之处

// //入队

// EnQueue(front, rear, 3);

// EnQueue(front, rear, 4);

// //出队

// DeQueue(front, rear);

// DeQueue(front, rear);

// DeQueue(front, rear);

//}

//

//int main(){

// LinkList front, rear;

// CircleQueue(front, rear);

// return 0;

//}

////第十四课 树与二叉树

//#include <stdio.h>

//#include <stdlib.h>

//

////二叉树定义

//typedef char BitElemType;

//typedef struct BiNode{

// BitElemType data;

// struct BiNode \*lchild;

// struct BiNode \*rchild;

//}BiNode, \*BiTree;

////辅助队列

//typedef struct tag{

// BiTree point; //以树结点为成员的队列

// struct tag \*pnext;

//}tag, \*ptag;

//

////层次建树 只限字符的写入

//void bitree\_create(BiTree &tree){

// BiTree tnew; //树中的新结点

// ptag pfront=NULL, prear=NULL; //辅助队列队头、队尾 //队头和当前树节点作用重复

// ptag pnew=NULL, pcur=NULL; //辅助队列新结点，当前树结点

// BitElemType input; //存放写入的数据

//

// while(scanf("%c", &input)){

// if(input=='\n'){

// break;

// }

// //calloc 空间为参数之积，并对空间初始化

// tnew=(BiTree)calloc(1, sizeof(BiNode));

// tnew->data=input;

// pnew=(ptag)calloc(1, sizeof(tag));

// pnew->point=tnew;

// //树根

// if(NULL==tree){

// tree=tnew;

// pfront=prear=pnew;

// pcur=pnew; //子树的父元素指定

// }else{

// //放入辅助队列

// prear->pnext=pnew;

// prear=pnew;

// //放入树

// if(NULL==pcur->point->lchild){

// pcur->point->lchild=tnew; //左子树

// }else if(NULL==pcur->point->rchild){

// pcur->point->rchild=tnew; //右子树

// pcur=pcur->pnext; //辅助队列指针

// }

// }

//

// }

//}

////前序遍历, 字符

//void bitree\_preorder(BiTree tree){

// if(tree){

// printf("%c", tree->data);

// bitree\_preorder(tree->lchild);

// bitree\_preorder(tree->rchild);

// }

//}

//

//

////代码复用-链队列

//typedef BiTree LiqElemType;

//typedef struct LinkNode{

// LiqElemType data;

// struct LinkNode \*next;

//}LinkNode;

//typedef struct{

// LinkNode \*front, \*rear;

//}LinkQueue;

////链队初始化 带头结点的链表实现

//void linkqueue\_init(LinkQueue &LQ){

// LQ.front=LQ.rear=(LinkNode\*)malloc(sizeof(LinkNode));

//// LQ.front=LQ.rear=(LinkNode\*)malloc(sizeof(LinkNode));

//// LQ.rear=(LinkNode\*)malloc(sizeof(LinkNode));

// LQ.front->next=NULL;

//// LQ.rear=LQ.front;

//}

////队列为空返回1

//bool linkqueue\_isempty(LinkQueue LQ){

//// if(LQ.rear==LQ.front){

//// return true;

//// }

//// return false;

// return LQ.rear==LQ.front;

//}

////入队

//void linkqueue\_en(LinkQueue &LQ, LiqElemType in){

//// if(0){

//// return false;

//// }

// LinkNode \*lq=(LinkNode\*)malloc(sizeof(LinkNode));

// lq->data=in;

// lq->next=NULL; //链队遍历结尾

// LQ.rear->next=lq;

// LQ.rear=lq;

//// return true;

//}

////出队

//bool linkqueue\_de(LinkQueue &LQ, LiqElemType &out){

// if(linkqueue\_isempty((LQ))){

// return false;

// }

// LinkNode \*lq=LQ.front->next;

// out=lq->data;

// LQ.front->next=lq->next; //lq->next的作用

// if(LQ.rear==lq){

// LQ.rear=LQ.front;

// }//链表只剩一个结点时，被删除前，需要改变rear

// free(lq);

// return true;

//}

//

////层序遍历 广度优先遍历

//void bitree\_leverorder(BiTree T){

// LinkQueue LQ;

// linkqueue\_init(LQ); //辅助链表初始化

// linkqueue\_en(LQ, T);

// LiqElemType pout;

//

// while(!linkqueue\_isempty(LQ)){

// linkqueue\_de(LQ, pout);

// putchar(pout->data);

// if(pout->lchild){

// linkqueue\_en(LQ, pout->lchild);

// }

// if(pout->rchild){

// linkqueue\_en(LQ, pout->rchild);

// }

// }

// printf("\n");

//}

//

//int main(){

// BiTree tree=NULL;

// bitree\_create(tree);

// printf("%c\n", tree->data);

//// printf("%c\n", tree->rchild->lchild->data);

//

// bitree\_preorder(tree);

// printf("\n");

// bitree\_leverorder(tree);

// printf("%d\n", '0');

// return 0;

//}

////第十四课 树与二叉树-实战

//#include <stdio.h>

//#include <stdlib.h>

//

////计算WPL(二叉树的带权路径长度)

//

////二叉树定义

//typedef int BitElemType;

//typedef struct BiNode{

// BitElemType weight;

// struct BiNode \*lchild;

// struct BiNode \*rchild;

//}BiNode, \*BiTree;

////辅助队列

//typedef struct tag{

// BiTree point; //以树结点为成员的队列

// struct tag \*pnext;

//}tag, \*ptag;

//

////层次建树 只限字符的写入

//void bitree\_create(BiTree &tree){

// BiTree tnew; //树中的新结点

// ptag pfront=NULL, prear=NULL; //辅助队列队头、队尾

// ptag pnew=NULL, pcur=NULL; //辅助队列新结点，当前树结点

// BitElemType input; //存放写入的数据

//

// while(scanf("%c", &input)){

// if(input=='\n'){

// break;

// }

// //calloc 空间为参数之积，并对空间初始化

// tnew=(BiTree)calloc(1, sizeof(BiNode));

// tnew->weight=input;

// pnew=(ptag)calloc(1, sizeof(tag));

// pnew->point=tnew;

// //树根

// if(NULL==tree){

// tree=tnew;

// pfront=prear=pnew;

// pcur=pnew; //子树的父元素指定

// }else{

// //放入辅助队列

// prear->pnext=pnew;

// prear=pnew;

// //放入树

// if(NULL==pcur->point->lchild){

// pcur->point->lchild=tnew; //左子树

// }else if(NULL==pcur->point->rchild){

// pcur->point->rchild=tnew; //右子树

// pcur=pcur->pnext; //辅助队列指针

// }

// }

//

// }

//}

//

////先序遍历计算wpl

//int wpl=0;

//void bitree\_wpl\_preorder(BiTree T, int deep){

//// static int wpl=0;//只会初始化一次

// if(T){

// if(NULL==T->lchild && NULL==T->rchild){

// wpl=wpl+T->weight\*deep;

// }

// bitree\_wpl\_preorder(T->lchild, deep+1);

// bitree\_wpl\_preorder(T->rchild, deep+1);

// }

//}

//

//int main(){

// BiTree tree;

// bitree\_create(tree);

// bitree\_wpl\_preorder(tree, 1);

// printf("wpl=%d\n", wpl);

//

// return 0;

//}

////第十五课 查找算法实战

////顺序查找、折半查找

//#include <stdio.h>

//#include <stdlib.h>

//#include <time.h>

//

//typedef int ElemType;

//typedef struct{

// ElemType \*element;

// int lenth;

//}SqTable;

////堆空间顺序表，需要初始化申请空间，c语言

////初始化-带哨兵

//void sqtable\_init2(SqTable \*ST, int len){

// ST->lenth=len+1;

// ST->element=(ElemType\*)malloc(sizeof(ElemType)\*ST->lenth);

// int i;

// srand(time(NULL)); //随机数生成

// //第零个是哨兵

// for(i=1; i<ST->lenth; i++){

// ST->element[i]=rand()%100; //生成0-99随机数

// }

//}

//////初始化，c++

////void sqtale\_init(SqTable &ST, int len){

//// ST.lenth=len+1;

//// ST.element=(ElemType\*)malloc(sizeof(ElemType)\*ST.lenth);

//// int i;

//// srand(time(NULL)); //随机数生成

//// //第零个是哨兵

//// for(i=1; i<ST.lenth; i++){

//// ST.element[i]=rand()%100; //生成0-99随机数

//// }

////初始化-不带哨兵

//void sqtable\_init(SqTable &ST, int len){

// ST.lenth=len;

// ST.element=(ElemType\*)malloc(sizeof(ElemType)\*ST.lenth);

// srand(time(NULL));

// int i;

// for(i=0; i<ST.lenth; i++){

// ST.element[i]=rand()%100;

// }

//}

////打印顺序表

////带哨兵

//void sqtable\_print2(SqTable ST){

// int i=0;

// for(i=1; i<ST.lenth; i++){

// printf("%3d", ST.element[i]);

// }

// printf("\n");

//}

////不带哨兵

//void sqtable\_print(SqTable ST){

// int i=0;

// for(i=0; i<ST.lenth; i++){

// printf("%3d", ST.element[i]);

// }

// printf("\n");

//}

//

////顺序查找，带哨兵、从后到前

//int sqtable\_search\_seq(SqTable ST, ElemType key){

// int i;

// ST.element[0]=key; //哨兵的作用

// for(i=ST.lenth; ST.element[i]!=key; i--){

// ;

// }

// return i;

//}

////qsort 参数函数

//int compare(const void \*left, const void \*right){//参数为任意两个元素的地址值

// return \*(ElemType\*)left-\*(ElemType\*)right; //从小到大排序

//// return -\*(ElemType\*)left+\*(ElemType\*)right; //从大到小

//}

////折半查找

//int sqtable\_search\_binary(SqTable ST, ElemType key){

// int head=0, tail=ST.lenth-1, mid; //front、back

// while(head<=tail){

// mid=(head+tail)/2;

// if(key<ST.element[mid]){

// tail=mid-1;

// }else if(key>ST.element[mid]){

// head=mid+1;

// }else{

// return mid;

// }

// }

// return -1;

//}

//

//#define Stnum 10

//int main(){

// SqTable ST;

// ElemType input=0;

// int pos;

//

// //顺序查找，采用带哨兵的init、print

// sqtable\_init2(&ST, Stnum);

// sqtable\_print2(ST);

// printf("sequence search key:");

// while(scanf("%d", &input)){

// pos=sqtable\_search\_seq(ST, input);

// if(pos){

// printf("sequence search pos is %d\n", pos);

// }else{

// printf("sequence search failed\n");

// }

// printf("sequence search key:");

// }

// free(ST.element);

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

// char c;

// scanf("%c", &c);

// //折半查找，使用对象：有序线性表

// //不带哨兵的init、print

// sqtable\_init(ST, Stnum);

// //排序函数qsort

// qsort(ST.element, ST.lenth, sizeof(ElemType), compare);

// sqtable\_print(ST);

// printf("binary search key:");

// while(scanf("%d", &input)){

// pos=sqtable\_search\_binary(ST, input);

// if(pos+1){

// printf("binary search pos is %d\n", pos);

// }else{

// printf("binary search failed\n");

// }

// printf("binary search key:");

// }

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

//

//

// return 0;

//}

////第十五课 查找算法实战

////二叉排序(查找)树

//#include <stdio.h>

//#include <stdlib.h>

//#include <time.h>

//

//typedef int ElemType;

//typedef struct BiNode{

// ElemType weight;

// struct BiNode \*lchild, \*rchild;

//}BiNode, \*BiTree;

////二叉排序树建树，相同元素不插入

////非递归

//int bitree\_init\_insert2(BiTree &BT, ElemType val){

// if(NULL==BT){

// BT=(BiTree)calloc(1, sizeof(BiNode));

// BT->weight=val;

// return 1;

// }

// BiTree bt=BT, parent; //bt用来遍历

// while(bt){

// parent=bt;

// if(val<bt->weight){

// bt=bt->lchild;

// }else if(val>bt->weight){

// bt=bt->rchild;

// }else{

// return 0; //排序树不能有重复元素

// }

// }

// bt=(BiTree)calloc(1, sizeof(BiNode));

// bt->weight=val;

// if(val<parent->weight){

// parent->lchild=bt;

// }else if(val>parent->weight){

// parent->rchild=bt;

// }else{

// return 0;

// }

// return 1;

//}

////带返回值

//int bitree\_init\_insert1(BiTree &BT, ElemType val){

// if(NULL==BT){

// BT=(BiTree)calloc(1, sizeof(BiNode));

//// BT->lchild=BT->rchild=NULL;

// BT->weight=val;

// return 1;

// }else if(val<BT->weight){

// return bitree\_init\_insert1(BT->lchild, val); //return 必须的

// }else if(val>BT->weight){

// return bitree\_init\_insert1(BT->rchild, val);

// }else{

// return 0;

// }

//}

////不带返回值

//void bitree\_init\_insert(BiTree &BT, ElemType val){

// if(NULL==BT){

// BT=(BiTree)calloc(1, sizeof(BiNode));

// // BT->lchild=BT->rchild=NULL;

// BT->weight=val;

// }else if(val<BT->weight){

// bitree\_init\_insert(BT->lchild, val); //return 必须的

// }else if(val>BT->weight){

// bitree\_init\_insert(BT->rchild, val);

// }else{

// ;

// }

//}

//void bitree\_init(BiTree &BT, ElemType \*arr, int num){

// //计算数据个数

// //元素逐个插入

// BT=NULL;

//// printf("%d\n", num);

// int i=0;

// for(i=0; i<num; i++){

// bitree\_init\_insert2(BT, arr[i]);

// }

//}

////二叉排序树查找

//BiTree bitree\_search(BiTree BT, ElemType key, BiTree &p){

// p=NULL;

// while(BT!=NULL && key!=BT->weight){

// p=BT;

// if(key<BT->weight){

// BT=BT->lchild;

// }else if(key>BT->weight){

// BT=BT->rchild;

// }

// }

// return BT;

//}

////二叉排序树中序遍历

//void bitree\_inorder(BiTree BT){

// if(BT){

// bitree\_inorder(BT->lchild);

// printf("%3d", BT->weight);

// bitree\_inorder(BT->rchild);

// }

//}

////二叉排序树，查找、递归

//void bitree\_delete(BiTree &root, ElemType del){

// if(NULL==root){

// return;

// }

// if(del<root->weight){

// bitree\_delete(root->lchild, del);

// }else if(del>root->weight){

// bitree\_delete(root->rchild, del);

// }else{ //开始删除结点

// if(NULL==root->lchild){

// BiTree temp=root; //用于释放结点

// root=root->rchild;

// free(temp);

// }else if(NULL==root->rchild){

// BiTree temp=root;

// root=root->lchild;

// free(temp);

// }else{

// //两边都不为空，采用左子树额最大数据（右子树的最小数据）代替

// BiTree tempnode=root->lchild; //遍历寻找

// while(tempnode->rchild!=NULL){

// tempnode=tempnode->rchild;

// }

// root->weight=tempnode->weight;

// //删除tempnode所在左子树最大结点

// bitree\_delete(root->lchild, root->weight);

// }

// }

//}

//

//int main(){

// BiTree BT=NULL; //树根

// ElemType arr[7]={54,20,66,40,28,70,58};

// //创建

// int num;

// num=sizeof(arr)/sizeof(ElemType);

// bitree\_init(BT, arr, num);

// //中序遍历

// bitree\_inorder(BT);

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

// //查找，考研要求查找父节点

// BiTree node, parent;

// node=bitree\_search(BT, 40, parent);

// if(node!=NULL){

// printf("bitree search value is %d\n", node->weight);

// }else{

// printf("bitree search failed\n");

// }

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

// //删除

// bitree\_delete(BT, 28);

// bitree\_inorder(BT);

//

// return 0;

//}

////第十五课 查找算法实战

////2014 42真题

////找出等长有序序列A和B，的合并中位数

//#include <stdio.h>

//

////基本思想

////1）分别找出A、B的中位数a,b

////2）a=b，则其为中位数

////3）a<b，舍弃A中较小一半及B中较大一半

////4）a>b，舍弃A中较大一半及B中较小一半

////5）重复以上步骤

////时间复杂度O(log2n), 空间复杂度O(1)

//

//int median\_search(int \*A, int \*B, int n){

// int start1=0, end1=n-1, start2=0, end2=n-1;

// int m1, m2;

// //循环至两数组都只剩一个

// while(start1!=end1 || start2!=end2){

// m1=(start1+end1)/2;

// m2=(start2+end2)/2;

// if(A[m1]==B[m2]){

// return A[m1];

// }else if(A[m1]<B[m2]){

// if((start1+end1)%2==0){

// //若元素为奇数个舍弃m1之前

// start1=m1;

// end2=m2;

// }else{

// //若元素为偶数个舍弃m1及其之前

// start1=m1+1;

// end2=m2;

// }

// }else{

// if((start2+end2)%2==0){

// start2=m2;

// end1=m1;

// }else{

// start2=m2+1;

// end1=m1;

// }

// }

// }

// //当剩余两个数不相等时，取较小数值

// return A[start1]<B[start2]?A[start1]:B[start2];

//}

//

//int main(){

// int A[]={11, 13, 15, 17, 19};

// int B[]={2, 4, 5, 6, 20};

// int mid=median\_search(A, B, 5);

// printf("median=%d\n", mid);

//

// return 0;

//}

////第十六课 排序算法

////交换类排序：冒泡排序、快速排序

//#include <stdio.h>

//#include <stdlib.h>

//#include <time.h>

//#include <string.h>

//

////堆空间顺序表（存储空间可控的数组）

//typedef int ElemType;

//typedef struct{

// ElemType \*element;

// int length;

//}SqTable;

//void sqtable\_init(SqTable &ST, int len){

// ST.length=len;

// ST.element=(ElemType\*)malloc(sizeof(ElemType)\*ST.length);

// int i;

// srand(time(NULL));

// for(i=0; i<ST.length; i++){

// ST.element[i]=rand()%100;

// }

//}

//void sqtable\_print(SqTable ST){

// int i;

// for(i=0; i<ST.length; i++){

// printf("%3d", ST.element[i]);

// }

// printf("\n");

//}

////交换顺序

//void swap(ElemType &a, ElemType &b){

// ElemType tmp=a;

// a=b;

// b=tmp;

//}

////冒泡排序，从大到小 O(n²),O(1)

//void sort\_bubble(ElemType \*B, int len){

// int i, j;

// bool flag;

// for(int i=0; i<len-1; i++){

// flag=true;

// for(j=len-1; j>i; j--){

// if(B[j]>B[j-1]){

// swap(B[j], B[j-1]);

// flag=false;

// }

// }

// if(flag){

// return;

// }

// }

//}

////快速排序，从小到大 O(nlogn),O(logn)

//int sort\_quick\_partition(ElemType \*Q, int front, int rear){

// //3,4,5,7,8,6,1,2,0,9 比3小的放左边、大的放右边

// ElemType pivot=Q[front];

// while(front<rear){

// while(front<rear && Q[rear]>=pivot){

// rear--;

// }

// Q[front]=Q[rear];

// while(front<rear && Q[front]<=pivot){

// front++;

// }

// Q[rear]=Q[front];

// }

// Q[front]=pivot;

// return front;

//}

//void sort\_quick(ElemType \*Q, int front, int rear){

// if(front<rear){

// int pivotpos=sort\_quick\_partition(Q, front, rear);

// sort\_quick(Q, front, pivotpos-1);

// sort\_quick(Q, pivotpos+1, rear);

// }

//}

//

//int main(){

// SqTable ST;

// sqtable\_init(ST, 10);

// sqtable\_print(ST);

//

// //固定值

// ElemType A[10]={3,4,5,7,8,6,1,2,0,9};

// memcpy(ST.element, A, sizeof(A)); //整型数组、浮点型数组使用memcpy

//

// //冒泡排序，时间复杂度为O(n²)

// sort\_bubble(ST.element, ST.length);

// sqtable\_print(ST);

//

// //快速排序

// sort\_quick(ST.element, 0, ST.length-1);

// sqtable\_print(ST);

//

// return 0;

//}

////第十六课 排序算法

////插入类排序：直接（简单）插入排序；选择类排序：简单选择排序

//#include <stdio.h>

//#include <stdlib.h>

//#include <time.h>

//#include <string.h>

//

////堆空间顺序表（存储空间可控的数组）

//typedef int ElemType;

//typedef struct{

// ElemType \*element;

// int length;

//}SqTable;

//void sqtable\_init(SqTable &ST, int len){

// ST.length=len;

// ST.element=(ElemType\*)malloc(sizeof(ElemType)\*ST.length);

// int i;

// srand(time(NULL));

// for(i=0; i<ST.length; i++){

// ST.element[i]=rand()%100;

// }

//}

//void sqtable\_print(SqTable ST){

// int i;

// for(i=0; i<ST.length; i++){

// printf("%3d", ST.element[i]);

// }

// printf("\n");

//}

////插入排序，从小到大 O(n²),O(1)

//void sort\_insert(ElemType \*I, int len){

// //将第一个数看成有序，后续数逐个插入其中

// int i,j;

// for(i=1; i<len; i++){

// int insertval=I[i];

// for(j=i-1; j>=0 && insertval<I[j]; j--){

// I[j+1]=I[j];

// }

// I[j+1]=insertval;

// }

//}

////简单选择排序，从大到小 O(n²),O(1)

//void swap(ElemType &a, ElemType &b){

// ElemType tmp=a;

// a=b;

// b=tmp;

//}

//void sort\_select(ElemType \*S, int len){

// int i, j;

// int max;

// for(i=0; i<len; i++){

// max=i;

// for(j=i+1; j<len; j++){

// if(S[j]>S[max]){

// max=j;

// }

// }

// swap(S[i], S[max]);

// }

//}

//

//int main(){

// SqTable ST;

// sqtable\_init(ST, 10);

// sqtable\_print(ST);

//

//// ElemType A[10]={3,4,5,7,8,6,1,2,0,9};

//// memcpy(ST.element, A, sizeof(A));

//

// //插入排序

// sort\_insert(ST.element, ST.length);

// sqtable\_print(ST);

//

// //选择排序

// sort\_select(ST.element, ST.length);

// sqtable\_print(ST);

//

// return 0;

//}

////第十七课 排序算法

////选择类排序：堆排序 归并排序

//#include <stdio.h>

//#include <stdlib.h>

//#include <time.h>

//#include <string.h>

//

//typedef int ElemType;

//typedef struct{

// ElemType \*element;

// int length;

//}SqTable;

//void sqtable\_init(SqTable &ST, int len){

// ST.length=len;

// ST.element=(ElemType\*)malloc(sizeof(ElemType)\*ST.length);

// srand(time(NULL));

// int i;

// for(i=0; i<ST.length; i++){

// ST.element[i]=rand()%100;

// }

//}

//void sqtable\_print(SqTable ST){

// int i=0;

// for(i=0; i<ST.length; i++){

// printf("%3d", ST.element[i]);

// }

// printf("\n");

//}

//void swap\_ElemType(ElemType &a, ElemType &b){

// ElemType tmp;

// tmp=a;

// a=b;

// b=tmp;

//}

//

////子树 排序为大根堆

//void sort\_heap\_max(ElemType \*h, int parent, int len){

// int child=parent\*2+1; //左子节点下标

// //循环或递归

//

// while(child<len){

// if(h[child]<h[child+1] && child+1<len){ //若len为偶数，h[child+1]可能为h[len+1]悬浮，会出问题

// child++;

// }

// if(h[child]>h[parent]){ //&& child<len，判断不该加在这

// swap\_ElemType(h[child], h[parent]);

// //发生交换后，子节点作为父节点循环

// parent=child;

// child=parent\*2+1;

// }else{

// break;

// }

// }

//}

////堆排序-从小到大，大根堆 sort\_heap\_max时间复杂度=树的高度log2n, 时间复杂度为你nlog2n

//void sort\_heap(ElemType \*H, int len){

// //从下到上调整为大根堆->最大值到达根节点

// for(int i=len/2-1; i>=0; i--){ //最后一个父节点为len/2-1

// sort\_heap\_max(H, i, len); //传入父节点下标

// }

// //交换根元素

// swap\_ElemType(H[0], H[len-1]);

// //剩余堆层次有序（即其他子树不需要重新排序），只需对H[0]树作大根堆排序

// for(int len\_unorder=len-1; len\_unorder>1; len\_unorder--){

// sort\_heap\_max(H, 0, len\_unorder);

// swap\_ElemType(H[0], H[len\_unorder-1]);

// }

//}

//

////归并排序-两两归并-递归，从小到大，非递归归并、非递归快排难度过高

//void sort\_merge\_together(ElemType \*M, int low, int mid, int high){

// static ElemType m[10]; //使得递归不增加m[n],空间复杂度为O(n)

// int i,j,k;

// for(k=low; k<=high; k++){

// m[k]=M[k];

// }

// for(i=low,j=mid+1,k=low; i<=mid&&j<=high; k++){//i为前半下标，j为后半下标，k为合并M下标

// if(m[i]<m[j]){

// M[k]=m[i++];

// }else{

// M[k]=m[j++];

// }

// }

// //将前/后部分剩余已排序元素排入

// while(i<=mid){

// M[k++]=m[i++];

// }

// while(j<=high){

// M[k++]=m[j++];

// }

//}

//void sort\_merge(ElemType \*M, int low, int high){

// if(low<high){

// int mid=(low+high)/2;

// sort\_merge(M, low, mid); //排序前一半

// sort\_merge(M, mid+1, high); //排序后一半

// sort\_merge\_together(M, low, mid, high); //排序好的合并

// }

//}

//int main(){

// SqTable ST;

// printf("-------heap sort-------\n");

// sqtable\_init(ST, 10);

// sqtable\_print(ST);

//// ElemType A[10]={3,87,2,93,78,56,61,38,12,40};

//// memcpy(ST.element, A, sizeof(A));

//// sqtable\_print(ST);

//

// sort\_heap(ST.element, ST.length);

// sqtable\_print(ST);

//

//

// printf("-------merge sort-------\n");

// sqtable\_init(ST, 10);

// sqtable\_print(ST);

// sort\_merge(ST.element, 0, ST.length-1);

// sqtable\_print(ST);

//

//

// return 0;

//}

//第十八课 考研真题

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

//2016 &43

//正整数集合A{ak|k>=0,k<n}，分成两个不想交子集A1、A2

//使其元素个数|n1-n2|最小且元素之和|S1-S2|最大

//使得|n1-n2|=0/1，返回|S1-S2|，平均时间复杂度O(n)=n+n/2+n/4+……

int half\_partition(int a[], int n){

// //函数内计算数组元素个数

// int i;

// for(i=0;a[i]!='\0';i++){

// ;

// }

// printf("函数内循环=%d\n", i);

// printf("函数内sizeof=%d\n", sizeof(a)/sizeof(a[0])); //计算指针大小

//快排思想，使得枢轴处于中间位置

int pivotvalue=0, front=0, front0=0, rear=n-1, rear0=n-1;

int flag=1; //标志位用以判断枢轴是否到中间位置 n/2，奇数个左侧少数

while(flag){

pivotvalue=a[front]; //枢轴选择

while(front<rear){ //根据枢轴划分数据

while(front<rear && pivotvalue<=a[rear]){ //至少一个包含等号，否则有相同数值会死循环

rear--;

}

if(front!=rear) a[front]=a[rear];

while(front<rear && pivotvalue>=a[front]){ //至少一个包含等号

front++;

}

if(front!=rear) a[rear]=a[front];

}

a[front]=pivotvalue;

if(front==(n/2-1)){ //判断枢轴位置，是==不是=

flag=0; //转换标志位，循环结束

}else if(front<n/2-1){ //对右半数据重新作枢轴划分

front0=++front; //等价于两个赋值语句，++需要写在前方

rear=rear0;

}else{ //对右半进行划分

front=front0;

rear0=--rear;

}

}

//求和

int s1=0, s2=0;

for(int i=0;i<n/2;i++){

s1+=a[i];

}

for(int i=n/2;i<=n-1;i++){

s2+=a[i];

}

return s2-s1;

}

//2022 &42

//n个数保存在一维数组M中，需要查找M中最小的十个数

//三个思路：（1）选择排序，每个元素升序插入代替最大元素，复杂度10n

//（2）堆排序，每个元素插入后代替堆顶，排序为大根堆

//（3）快速排序，只排序一半直到缩小到10个

//堆排序实现

typedef int ElemType;

//空间可控数组，附带数组长度

typedef struct{

ElemType \*element;

int length;

}SqTabel;

//可控数组初始化，随机生成len个100内的数

void sqtable\_init(SqTabel &ST, int len){

ST.length=len;

ST.element=(ElemType\*)malloc(sizeof(ElemType)\*ST.length);

srand(time(NULL));

for(int i=0;i<ST.length;i++){

ST.element[i]=rand()%100;

}

}

//可控数组打印，前n个

void sqtable\_print(SqTabel ST, int len){

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

printf("%3d", ST.element[i]);

}

printf("\n");

}

void swap\_ElemType(ElemType &a, ElemType &b){

ElemType tmp;

tmp=a;

a=b;

b=tmp;

}

//将前len个元素中，dad为父节点的小子树调整为大根堆

void sort\_heap\_max(ElemType \*h, int dad, int len){

int son=2\*dad+1; //左子结点下标

while(son+1<=len){

if(son+1<=len && h[son]<h[son+1]){//若有右子节点则比较大小

son++;

}

if(h[son]>h[dad]){

swap\_ElemType(h[son], h[dad]);

dad=son; //子节点作为大根堆判断下一棵是否符合

son=2\*dad+1;

}else{

break;

}

}

}

//在原有数组内排序

void sort\_heap(SqTabel ST, int len){

//先将前len个元素调整为大根堆，从最后一个父节点开始

for(int i=len/2-1;i>=0;i--){ //最后一个父节点为element[4]

sort\_heap\_max(ST.element, i, len);

}

for(int i=len;i<ST.length;i++){

if(ST.element[i]<ST.element[0]){

ST.element[0]=ST.element[i];

sort\_heap\_max(ST.element, 0, len);

}

}

}

int main(){

//快排思想解决2016 &43

int A[11]={4, 1, 12, 18, 7, 13, 18, 16, 2, 15};

int DifferenceValue;

DifferenceValue=half\_partition(A, 10);

// //函数外计算数组元素个数

// int i;

// for(i=0;A[i]!='\0';i++){

// ;

// }

// printf("函数外循环=%d\n", i);

// printf("函数外sizeof%d\n", sizeof(A)/sizeof(A[0]));

printf("|S1-S2|=%d\n", DifferenceValue);

//堆排序 2022 &42

SqTabel ST;

sqtable\_init(ST, 100);

// ElemType B[10]={3,1,2,4,7,6,9,8,5,3};

// printf("sizeof(B)=%d\n", sizeof(B));

// memcpy(ST.element, B, sizeof(B));

sort\_heap(ST, 10);

sqtable\_print(ST, 10);

return 0;

}

////第十九课 C语言语法

//#include <stdio.h>

//#include <stdlib.h>

//

//int main(){

// //左移、右移

// short i=5;

// short j;

// j=i<<1;

// printf("i<<1=%d\n", j);

// i=-7;

// j=i>>1;

// printf("i>>1=%d\n", j);

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

// //无符号数

// i=0x8011;

// unsigned short s=0x8011; //0x-十六进制

// unsigned short r=0;

// j=i>>1; //有符号数右移 i=-32751

// r=s>>1; //无符号数右移 s=32785

// printf("j=%d, r=%u\n", j, r);

// //按位

// i=5, j=7;

// printf("i&j=%d\n", i&j);

// printf("i|j=%d\n", i|j);

// printf("i^j=%d\n", i^j);

// printf("~i=%d\n", ~i);

// //^异或满足交换律

// int arr[7]={1,1,8,5,3,5,8}; //找出只出现一次的一个数

// int result=0;

// for(int i=0;i<7;i++){

// result^=arr[i];

// }

// printf("result=%d\n", result);

//

// //二维数组

// int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};

// printf("sizeof(a)=%d\n", sizeof(a));

//

// //switch

// int mon,year;

// while(scanf("%d %d", &year, &mon)){

// switch (mon) {

// case 2:

// printf("month%d has %d days\n", mon, 28+(year%4==0&&year%100!=0 || year%400==0));

// break;

// case 1: case 3: case 5: case 7: case 8: case 10: case 12:

// printf("month%d has %d days\n", mon, 31);

// break;

// case 4: case 6: case 9: case 11:

// printf("month%d has %d days\n", mon, 30);

// break;

// default:

// printf("error month\n");

// break;

// }

// }

//

// //二级指针

//

// return 0;

//}

////第二十课 数据的机器级表示

//#include <stdio.h>

//#include <stdlib.h>

//

//int main(){

// //整型溢出

// short a=32767;

// short b=a+1;

// int c=a+1;

// printf("%d\n", b);

// printf("%d\n", c);

//

// unsigned short d=0x8056; //32854

// b=0x8056;

// printf("%d\n", b);

// printf("%u\n", d);

//

// //浮点数精度丢失

// float a1=1.6777216; //1.6777216e7是边界

// float b1;

// b1=a1+2; //精度丢失 12345678920

// printf("b1=%f\n", a1);

//

// return 0;

//}

////第二十一课 汇编语言入门

//#include <stdio.h>

//

//int main(){

// int i=0;

// printf("%d\n", i);

//

// return 0;

//}

////第二十二课 汇编语言实战

//#include <stdio.h>

//

//int main(){

// unsigned int x=134;

// unsigned int y=246;

// int m=x;

// int n=y;

// unsigned int z1=x-y;

// unsigned int z2=x+y;

// printf("%d %d\n", m, n);

// printf("%u %u\n", z1, z2);

//

// return 0;

//}

////第二十三课 文件操作

//#include <stdio.h>

//#include <stdlib.h>

//#include <string.h>

//

//int main(){

// FILE \*fp;

// //fopen

// fp=fopen("test.txt", "r+");

// if(!fp){

// perror("fopen");

// fclose(fp);

// return -1;

// }

//

//// //fgetc、fputc

//// char c;

//// while((c=fgetc(fp))!=EOF){

//// printf("%c", c);

//// }

//// c=fputc('H', fp);

//// if(-1==c){

//// perror("fput");

//// fclose(fp);

//// return -1;

//// }

//

// //fwrite、fread 读写不能同时进行，因为ptr指针位置

// char buf[20]="hello\nworld";

// int ret, len=0;

// long pos;

// len=strlen(buf);

// ret=fwrite(buf, sizeof(char), len, fp); //写入内容

// ret=fseek(fp, -5, SEEK\_CUR); //当前位置向前偏移5字符

// if(ret!=0){

// perror("fseek");

// fclose(fp);

// return -1;

// }

// pos=ftell(fp);

// printf("Current pos = %ld\n", pos);

// memset(buf, 0, sizeof(buf)); //初始化buf

// ret=fread(buf, sizeof(char), sizeof(buf), fp);

// printf("%s\n", buf);

//

//// char buf1[20]={0};

//// ret=fread(buf1, sizeof(char), sizeof(buf1), fp);

//// printf("%s\n", buf1);

//

// //fgets、fputs

//

// //fseek、ftell

//

// fclose(fp);

// return 0;

//}