#include <stdio.h>

#include <stdlib.h>

#define N 1000

int Arr[N] = { 0 };

int iNumber = 0; //记录元素个数

//=====================================================

//主系统

void ShowinStructions();

//输入数据

void InputData(int \*arr,int\* number);

//输出数据

void OutputData(int\* arr, int\* number);

void swap(int\* a, int\* b); //数值交换

//=====================================================

//排序系统

void showSortStruct();

void DataSort();

//冒泡排序

void popo(int\* x, int n);

//选择排序

void selectSort(int\* arr, int num);

//插入排序

void insertSort(int\* arr, int num);

//希尔排序

void shellSort(int\* arr, int num);

//归并排序

void mergeSort(int\* arr, int num);

//快速排序

void quicksort(int left, int right);

//堆排序

void heap(int\* x, int n);

//基数排序

void rads(int\* x, int n, int d, int k);

//=====================================================

//查找系统

void showFindStruct();

void DataFind();

int main(void)

{

int i = 0; //选择模式

int flag = 1; //为了可以可控退出循环

ShowinStructions();

while (flag)

{

printf("请选择模式(9.显示指令)：\n");

char str[5] = { 0 };

scanf("%s", str);

if (str[0] == '0')

{

system("cls");

printf("退出系统\n");

flag = 0;

break;

}

i = atoi(str); //将字符串转换为数字，转换成功则等于对应数字，否则失败i = 0

if (i != 0)

{

switch (i)

{

case 1:

//输入数据

InputData(Arr, &iNumber);

break;

case 2:

system("cls");

printf("进入排序系统\n");

Sleep(500);

system("cls");

DataSort();

break;

case 3:

system("cls");

printf("进入查找系统\n");

Sleep(500);

system("cls");

DataFind();

break;

case 4:

//重置数据

break;

case 5:

//输出数据

OutputData(Arr, &iNumber);

break;

case 8:

//清空控制台

system("cls");

break;

case 9:

//展示指令

ShowinStructions();

break;

default:

printf("指令错误错误！\n");

break;

}

}

else

{

printf("输入格式错误！\n");

}

}

system("pause>0");

return 0;

}

//主系统指令

void ShowinStructions()

{

printf("\t指令如下：\n");

printf("\t1、输入数据\n");

printf("\t2、数据排序\n");

printf("\t3、数据查找\n");

printf("\t4、重置数据\n");

printf("\t5、输出数据\n");

printf("\t8、清空页面\n");

printf("\t9、显示指令\n");

printf("\t0、退出系统\n");

}//输出数据

void OutputData(int\* arr, int\* number)

{

printf("数据如下：\n");

for (int i = 0; i < (\*number); i++)

{

printf("%d ", \*(arr + i));

}

putchar('\n');

}

//输入数据

void InputData(int\* arr, int\* number)

{

printf("请输入数据个数\n");

int j = 0;

scanf("%d", &j);

int n = (\*number);

for (int i = (\*number); i < j + n; i++)

{

printf("第%d个数据：", i + 1);

scanf("%d", arr + i);

(\*number)++;

}

}

//查找系统=====================================================================

//查找框架

void DataFind()

{

showFindStruct();

int i = 0; //选择模式

int flag = 1; //为了可以可控退出循环

while (flag)

{

printf("请选择查找(9.显示指令)：\n");

char str[5] = { 0 };

scanf("%s", str);

if (str[0] == '0')

{

system("cls");

printf("退出查找\n");

Sleep(500);

printf("进入主系统\n");

Sleep(500);

system("cls");

ShowinStructions();

flag = 0;

break;

}

i = atoi(str); //将字符串转换为数字，转换成功则等于对应数字，否则失败i = 0

if (i != 0)

{

switch (i)

{

case 1:

//顺序查找

break;

case 2:

//二分查找

break;

case 3:

//树表查找

break;

case 4:

//分块查找

break;

case 5:

//哈希查找

break;

case 6:

//插值查找

break;

case 7:

//斐波那契查找

break;

case 8:

break;

case 9:

showFindStruct();

break;

default:

printf("指令错误错误！\n");

break;

}

}

else

{

printf("输入格式错误！\n");

}

}

}

//查找指令

void showFindStruct()

{

printf("\t查找如下：\n");

printf("\t1、顺序查找\n");

printf("\t2、二分查找\n");

printf("\t3、树表查找\n");

printf("\t4、分块查找\n");

printf("\t5、哈希查找\n");

printf("\t6、插值查找\n");

printf("\t7、斐波那契查找\n");

printf("\t9、展示指令\n");

printf("\t0、退出查找\n");

}

//排序系统=====================================================================

//排序框架

void DataSort()

{

showSortStruct();

int i = 0; //选择模式

int flag = 1; //为了可以可控退出循环

while (flag)

{

printf("请选择排序(9.显示指令)：\n");

char str[5] = { 0 };

scanf("%s", str);

if (str[0] == '0')

{

system("cls");

printf("退出排序\n");

Sleep(500);

printf("进入主系统\n");

Sleep(500);

system("cls");

ShowinStructions();

flag = 0;

break;

}

i = atoi(str); //将字符串转换为数字，转换成功则等于对应数字，否则失败i = 0

if (i != 0)

{

switch (i)

{

case 1:

//冒泡排序

popo(Arr, iNumber);

OutputData(Arr, &iNumber);

break;

case 2:

//选择排序

selectSort(Arr,iNumber);

OutputData(Arr, &iNumber);

break;

case 3:

//插入排序

insertSort(Arr,iNumber);

OutputData(Arr, &iNumber);

break;

case 4:

//快速排序

quicksort(0,iNumber);

OutputData(Arr, &iNumber);

break;

case 5:

//希尔排序

shellSort(Arr,iNumber);

OutputData(Arr, &iNumber);

break;

case 6:

//归并排序

mergeSort(Arr, iNumber);

OutputData(Arr, &iNumber);

break;

case 7:

//基数排序

rads(Arr, iNumber, iNumber, 10);

OutputData(Arr, &iNumber);

break;

case 8:

//堆排序

heap(Arr, iNumber);

OutputData(Arr, &iNumber);

break;

case 9:

//展示指令

showSortStruct();

break;

case 10:

system("cls");

break;

default:

printf("指令错误错误！\n");

break;

}

}

else

{

printf("输入格式错误！\n");

}

}

}

//排序指令

void showSortStruct()

{

printf("\t排序如下：\n");

printf("\t1、冒泡排序\n");

printf("\t2、选择排序\n");

printf("\t3、插入排序\n");

printf("\t4、快速排序\n");

printf("\t5、希尔排序\n");

printf("\t6、归并排序\n");

printf("\t7、基数排序\n");

printf("\t8、堆排序\n");

printf("\t9、展示指令\n");

printf("\t10、清空控制台\n");

printf("\t0、退出排序\n");

}

//冒泡排序

void popo(int\* x, int n)

{

int head, tail, i, j, t, flag;

head = 0;

tail = n - 1;

flag = 1;

while ((head < tail) && (flag == 1))

{

flag = 0;

j = tail;

for (i = head; i < j; i++) /\* 从头向后扫描\*/

if (x[i] > x[i + 1])

{

t = x[i];

x[i] = x[i + 1];

x[i + 1] = t;

tail = i; /\* 当前交换到的最大值位置\*/

flag = 1; /\* 若有交换，则说明还未完全有序\*/

}

j = head;

for (i = tail; i > j; i--) /\* 从后向头扫描\*/

if (x[i - 1] > x[i])

{

t = x[i];

x[i] = x[i - 1];

x[i - 1] = t;

head = i; /\* 交换到的最小值位置\*/

flag = 1; /\* 若有交换，则说明还未完全有序\*/

}

}

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

{

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

}

printf("\n");

return;

}

//选择排序

void selectSort(int\* arr, int num)

{

int min, i, j;

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

{

min = i;

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

{

if (arr[j] < arr[min]) {

min = j;

}

}

if (min != i)

{

swap(&arr[min], &arr[i]);

}

}

}

//交换算法

void swap(int\* a, int\* b)

{

int tmp = \*a;

\*a = \*b;

\*b = tmp;

}

//插入排序

void insertSort(int\* arr, int num)

{

int i, j, tmp;

for (i = 1; i < num; ++i)

{

tmp = arr[i];

for (j = i; j > 0 && arr[j - 1] > tmp; --j)

{

arr[j] = arr[j - 1];

}

arr[j] = tmp;

}

}

//希尔排序

void shellSort(int\* arr, int num)

{

int gap, i, j, temp;

for (gap = num >> 1; gap > 0; gap = gap >>= 1)

for (i = gap; i < num; i++)

{

temp = arr[i];

for (j = i - gap; j >= 0 && arr[j] > temp; j -= gap)

{

arr[j + gap] = arr[j];

}

arr[j + gap] = temp;

}

}

//归并排序

void mergeSort(int\* arr, int num) {

int\* a = arr;

int\* b = (int\*)malloc(num \* sizeof(int));

int seg, start;

for (seg = 1; seg < num; seg += seg) {

for (start = 0; start < num; start += seg + seg) {

int low = start, mid = min(start + seg, num), high = min(start + seg + seg, num);

int k = low;

int start1 = low, end1 = mid;

int start2 = mid, end2 = high;

while (start1 < end1 && start2 < end2) {

b[k++] = a[start1] < a[start2] ? a[start1++] : a[start2++];

}

while (start1 < end1) {

b[k++] = a[start1++];

}

while (start2 < end2) {

b[k++] = a[start2++];

}

}

int\* temp = a;

a = b;

b = temp;

}

if (a != arr) {

int i;

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

b[i] = a[i];

b = a;

}

free(b);

}

//快速排序

void quicksort(int left, int right) { //left 左，right 右

int i, j, t, temp;

if (left > right)

return;

temp = Arr[left]; //temp中存的就是基准数

i = left;

j = right;

while (i != j) { //顺序很重要，要先从右边开始找

while (Arr[j] >= temp && i < j)

j--;

while (Arr[i] <= temp && i < j)//再找右边的

i++;

if (i < j)//交换两个数在数组中的位置

{

t = Arr[i];

Arr[i] = Arr[j];

Arr[j] = t;

}

}

//最终将基准数归位

Arr[left] = Arr[i];

Arr[i] = temp;

quicksort(left, i - 1);//继续处理左边的，这里是一个递归的过程

quicksort(i + 1, right);//继续处理右边的 ，这里是一个递归的过程

}

//堆排序

void heap(int\* x, int n)

{

int i, t;

static void sift();

for (i = n / 2 - 1; i >= 0; i--) /\* 构建初始堆\*/

sift(x, n, i);

for (i = 1; i < n; i++) /\* 依次弹出n-1个根结点，并维护堆\*/

{

t = x[0]; /\* x[0]就是根，也是当前最大值\*/

x[0] = x[n - i];

x[n - i] = t;

sift(x, n - i, 0); /\* 筛x[0]\*/

}

return;

}

static void sift(x, n, i) /\* 对x[n]数组中x[i]元素进行筛运算，\*/

int\* x, n, i; /\* 假定以x[i+1]到x[n-1]的每个元素为根的子树已经成为堆\*/

{

int j, t;

t = x[i]; /\* 待筛结点的值\*/

j = 2 \* i + 1; /\* x[j]是x[i]的左孩子\*/

while (j < n)

{

if ((j < n - 1) && (x[j] < x[j + 1])) /\* 放入与较大的右孩子一边\*/

j++;

if (t < x[j])

{

x[i] = x[j]; /\* 较大的元素放在x[i]中\*/

i = j; /\* 同时i下沉一层\*/

j = 2 \* i + 1; /\* 在求i的左孩子\*/

}

else

break; /\* 此时或已经到底，或比左孩子和右孩子都大\*/

}

x[i] = t;

}

//基数排序

void rads(int\* x, int n, int d, int k)

{

int j, m, k1, \* a, \* y, flag;

void rads\_cout();

a = (int\*)malloc(n \* sizeof(int)); /\* a是存放一位数字的数组\*/

y = (int\*)malloc(n \* sizeof(int)); /\* y是用于存放中间变量的空间\*/

flag = 0;

k1 = 1;

for (m = 0; m < d; m++) /\* 分别按d位进行排序\*/

{

if (flag == 0)

{

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

{

a[j] = x[j] / k1; /\* 得到了第m位\*/

a[j] = a[j] % k;

}

rads\_cout(a, x, y, n, k);

}

else

{

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

{

a[j] = y[j] / k1; /\* 得到了第m位\*/

a[j] = a[j] % k;

}

rads\_cout(a, y, x, n, k);

}

flag = 1 - flag;

k1 = k1 \* k;

}

if (flag == 1) /\* 此时排序后数据存放在y中\*/

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

x[j] = y[j];

free(a);

free(y);

return;

}

void rads\_cout(int\* a, int\* x, int\* y, int n, int k) /\* 将x按a排序，结果放在y中\*/

{

int i, j, \* c;

c = (int\*)malloc((k + 1) \* sizeof(int));

for (i = 0; i <= k; i++) /\* 设置初值\*/

c[i] = 0;

for (i = 0; i < n; i++) /\* 统计个数\*/

{

j = a[i];

c[j] = c[j] + 1;

}

for (i = 1; i <= k; i++) /\* 计数，用于计算位置\*/

c[i] = c[i] + c[i - 1];

for (i = n - 1; i >= 0; i--) /\* 直接各归其位\*/

{

j = a[i];

y[c[j] - 1] = x[i];

c[j] = c[j] - 1;

}

free(c);

return;

}