第六次作业参考答案

钟文杰　　2025.11.4

作业 11

P157 T13

基于例 4.3-14 编写程序，产生 100 个和 10000 个 $[0, 20)$ 区间的整数，然后统计并打印 $0 \sim 19$ 在其中出现次数的占比（概率）。

作业反馈：大家这题做的都没什么问题。抱有“为什么要在生成随机数前用调用 time 函数生成种子”及类似困惑的同学可以自行翻阅课本上随机数库函数的相关内容，这里就不赘述了。

参考代码：

```
//思路：边生成随机数边统计，就不用存储10000个数了。
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
int main(){
    int a[25]={0},b[25]={0};
    srand((unsigned) time(NULL));
    for(int i=0;i<100;i++) a[rand()%20+1]++;
    for(int i=0;i<10000;i++) b[rand()%20+1]++;
    puts("随机生成100次后出现的次数占比：");
    for(int i=1;i<=20;i++){
        printf("%d:%.2f\t\t",i,a[i]/100.0);
        if(i%5==0) printf("\n");
    }
    puts("随机生成10000次后出现的次数占比：");
    for(int i=1;i<=20;i++){
        printf("%d:%.4f\t",i,b[i]/10000.0);
        if(i%5==0) printf("\n");
    }
    return 0;
}
```

Note：想要生成真随机数，仅凭计算机和编写好的程序是做不到的，必须要借助现实世界中的一些物理过程。

P157 T14

上周我把这道题放进答案里的时候没看到这周有这道题，早知道上周就不用放这道题了 *www*

设计与实现接收数组并对其中的数据进行降序排序的插入法排序函数。

作业反馈：有一部分同学没有按照题目要求将插入排序写成单独一个函数，还有同学用的不是插入排序，这类错误这次都扣了分。请务必重视题目要求！

参考代码：

```
//思路：利用地址形参实现函数直接对待排序数组做改动
#include<stdio.h>
void sort(int* a,int n){//没有必要返回任何值的函数就设置为void类型
    for(int i=1;i<n;i++){
        int t=a[i],j=0;
        for(;j<i;j++){
            if(a[j]<t) break;
        }//这个循环也可以写成for(;a[j]<t;j++); 想想为什么?
        for(int k=i-1;k>=j;--k){//写循环一定要注意终止条件
            a[k+1]=a[k];
        }
        a[j]=t;
    }
    return;
}
int main(){
    int a[10]={1,2,3,99,45,32,193};//图方便直接内置值
    sort(a,7);
    for(int i=0;i<6;++i) printf("%d ",a[i]);
    printf("%d\n",a[6]);//最后一个值后面不输出空格而是输出回车
    //适用于要求数值后不能有多余空格、回车的OJ
    return 0;
}
```

note: 有同学对程序的时间复杂度这一概念不大了解，这里作简单的介绍，有兴趣的同学可以进一步与助教交流或查询网络资料。

时间复杂度是算法执行时间关于问题规模的函数，它体现了该算法处理问题耗时随问题规模变大的变化趋势。计算（或者说估算，因为我们后面就会看到我们实际上并不关注也没法得到这个函数的具体表达式，我们只考虑最大的若干项）时间复杂度基于以下假设：

0. 程序是可以在有限时间内结束的。（序号 0 表示这是一切的前提，后面的 123... 都是并列关系，但这条跟他们不并列）

1. 基础运算（如加减乘除，关系运算）消耗的时间是常数量级，即我们忽略参与运算的数字变大导致的基础运算用时延长。

2. 程序能够处理大规模的问题，因而常数相对于问题的规模 n 来说是可以忽略的。（比如我们认为 $2n$ 和 n 是一样大的）

3. 一般来说时间复杂度指的是算法在不同情况下的平均耗时，如果专门考虑最优（或最差）情况我们会专门称其为最优（最差）情况下的时间复杂度。

以上述的程序为例，赋 n 个初值（或者读入 n 个值）需要 n 次运算，因而时间复杂度为 $O(n)$（这个记号表示处理问题的时间是 n 量级）而后续的排序过程需要对 n 个值做双循环，第一个循环从 1 到 $n-1$, 第二个从 0 到 $i-1$，又从 $i-1$ 到某个 j，因而最多需要 $2*(1+2+...+n)=n*(n+1)$ 次操作。但我们前面假设，常数对于 n 来说是可以忽略的，因此这里的时间复杂度是 $O(n^2)$。后面输出的时间复杂度同样为 $O(n)$, 总计时间复杂度是 $O(n^2+2n)$. 但我们只考虑最大的项，从而总时间复杂度是 $O(n^2)$.

我们常说的优化算法一般都是在减小算法的时间复杂度，程序本身的长度和运行时占用的内存都不那么重要。

P157 T16

实现优化的冒泡排序，并构造一个主函数，从预先写好数据的文本文件中读数据到数组中，调用该函数中打印排序结果。

作业反馈：书上 P139 页末尾有说过怎么优化冒泡排序，因此不作任何优化的程序会被扣分。这次仍然没有写文件读写的也会被扣分。

参考代码：

```
//思路：每次循环标记最后一次发生交换的位置，下一次循环只需运行到那里
#include<stdio.h>
#include<stdlib.h>
void sort(int* a,int n){
    int tmp=0,book=1;//book标记一次循环中是否有发生交换
    for(int i=n-1,k=0;book==1;){//没有发生任何交换时结束循环
        book=0;
```

```
        k=i;
        for(int j=0;j<k;j++) if(a[j]>a[j+1]){
            tmp=a[j];
            a[j]=a[j+1];
            a[j+1]=tmp;
            i=j;//在整个循环结束后，i会停在最后一次发生交换的地方
            book=1;
        }//现在i后面的数据都已经排好序了
    }
    return;
}
int main(){
    int a[105]={0},n=0;
    FILE* fp=fopen("P157_16_in.txt","r");
    if(fp==NULL){
        puts("Error!");
        exit(-1);
    }
    fscanf(fp,"%d",&n);
    for(int i=0;i<n;i++) fscanf(fp,"%d",&a[i]);
    fclose(fp);
    fp=fopen("P157_16_out.txt","w+");
    if(fp==NULL){
        puts("Error!");
        exit(-1);
    }
    sort(a,n);
    for(int i=0;i<n;i++) fprintf(fp,"%d ",a[i]);
    fclose(fp);
    return 0;
}
```

note: 冒泡排序其实几乎只有这一种优化方式。按照上面说过的时间复杂度的定义，一些小量级操作量的优化实际上几乎没什么效果，这类优化在相关圈子里俗称“卡常”，算是没什么必要了解的奇技淫巧。

P157 T22

编写实现如下功能的函数：

(1) 信息录入函数：输入 10 名学生的学号和姓名。

(2) 排序函数：实现按学号从小到大排序，姓名也随之调整。

(3) 查找函数：根据输入的同学的姓名来查找，找到的话就打印其学号。

(4) 主函数：调用以上三个函数，实现简易学生信息管理系统。自定义数据结构。

作业反馈：大家基本上功能实现的都没什么问题。

参考代码：

```
//思路：利用结构体按要求实现功能
#include<stdio.h>
struct student{
    int No;
    char Name[20];
};
void input(struct student* stu,int n){
    for(int i=0;i<n;i++) scanf("%d %s",&stu[i].No,stu[i].Name);
    return;
}
void sort(struct student* stu,int n){
    struct student tmp={0};
    int book=1;
    for(int i=n-1,k=0;book==1;){
        book=0;
        k=i;
        for(int j=0;j<k;j++) if(stu[j].No>stu[j+1].No){
            tmp=stu[j];
            stu[j]=stu[j+1];
            stu[j+1]=tmp;
            i=j;
            book=1;
        }
    }
```

```
    return;
}
int strcmp(char* str1,char* str2){
    int i=0;
    for(;str1[i]!='\0';i++) if(str1[i]!=str2[i]) return 0;
    if(str2[i]=='\0') return 1;
    else return 0;
}
int find(char* Tofind,struct student* stu,int n){
    for(int i=0;i<n;i++)
        if(strcmp(Tofind,stu[i].Name))
            return stu[i].No;
    return -1;
}
int main(){
    int k=0;
    char Tofind[20]={0};
    struct student stu[15]={0};
    input(stu,10);
    sort(stu,10);
    scanf("%s",Tofind);
    k=find(Tofind,stu,10);
    if(k==-1) puts("没有找到这个学生。");
    else printf("这个学生的学号是: %d\n",k);
    return 0;
}
```

note: 同类结构体之间是可以直接进行赋值的，因此不必对结构体中的每个成员分别赋值。

P157 T23

利用递归算法，把整数 n 的每一位数分解，并从高位到低打印出来，一行输出一个数。（提示：应考虑输入为负数的情况，如果为负数，就先把负号打印，再处理整数部分。）

作业反馈：即使课本已经提示了要处理负数情形，仍然有相当一部分同学没有进行对应的处理，下次见到这种错我就要下狠手多扣点分了。还有一部分同学

没有按照题目要求利用递归算法进行处理，也同样是拿不到满分的。

参考代码：

```
//思路：如题目所述，记得处理符号
#include<stdio.h>
void partition(int n){
    if(n==0) return;
    partition(n/10);
    printf("%d\n",n%10);
    return;
}
int main(){
    int n=0;
    scanf("%d",&n);
    if(n<0){
        puts("-");
        n=-n;
    }
    if(n==0) puts("0");
    partition(n);
    return 0;
}
```

P157 T25

将 4.4.2 节的排序与查找练习程序改成有如下菜单的程序。

(1) 选择产生数据的方式：1. 随机产生；2. 产生斐波那契数列；3. 手工输入。

(2) 选择排序方式：1. 升序；2. 降序。

(3) 输入查询的数据。

(4) 退出。

将程序保持在循环状态，直至选择“4”后结束程序的运行。注意未产生数据就排序、菜单选项输入异常、数据输入异常的情况。

作业反馈：没什么人选做这道题，但选做这道题的同学的程序写的都没什么问题，想来这道题对于大家而言应该并不困难，只是确实相当繁琐。

参考代码：

```
//思路：按照题目要求逐一实现功能。程序很长，需要充分了解程序的结构。
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<windows.h>
//这个库提供一些windows系统相关的操作，有兴趣的同学可以自行了解
void printmenu(void){
    system("cls");
    //将终端清空，避免程序运行几次过后输出过的东西太多屏幕显示不完
    puts("请选择要进行的操作：（输入对应序号）");
    puts("(1) 生成/输入数据");
    puts("(2) 对数据进行排序");
    puts("(3) 查询数据");
    puts("(4) 退出程序");
    return;
}
void printArray(int* a,int n){
    for(int i=0;i<n-1;i++) printf("%d ",a[i]);
    printf("%d\n",a[n-1]);
    system("pause");//这行代码可以让程序暂停，方便用户查看输出的信息
    //否则立马就会清空终端重新输出菜单，用户看不到输出
    return;
}
int input(int* a,int m){
    int n=0,mode=0;
    puts("要生成/输入多少个数据？");
    scanf("%d",&n);
    if(n>m){//记得限制数据个数不能超过数组大小
        puts("数据太多了！");
        system("pause");
        return -1;
    }
    printf("如何生成数据？");
    printf("1-随机生成；2-生成斐波那契数列；3-手动输入\n");
    scanf("%d",&mode);
```

```
    switch(mode){
        case 1:{
            for(int i=0;i<n;i++) a[i]=rand()%1000;
            puts("生成的随机数据如下：");
            printArray(a,n);
            break;
        }
        case 2:{
            a[0]=1;
            if(n>1) a[1]=1;
            for(int i=2;i<n;i++) a[i]=a[i-1]+a[i-2];
            puts("生成的斐波那契数列如下：");
            printArray(a,n);
            break;
        }
        case 3:{
            for(int i=0;i<n;i++) scanf("%d",&a[i]);
            break;
        }
        default:{
            puts("无效操作！");
            system("pause");
            return -1;
        }
    }
    return n;
}
void sort(int* a,int n){
    int tmp=0,book=1,t=0;//t标记排序方式，t=1为升序，t=0为降序
    puts("请选择排序方式：1-升序；2-降序");
    scanf("%d",&t);
    t=2-t;
    for(int i=n-1,k=0;book==1;){
        book=0;
        k=i;
```

```
        for(int j=0;j<k;j++) if(t?(a[j]>a[j+1]):(a[j]<a[j+1])){
            tmp=a[j];
            a[j]=a[j+1];
            a[j+1]=tmp;
            i=j;
            book=1;
        }
    }
    return;
}
int Search_binary(int* a,int n,int Tofind){
    int low=0,high=n-1,mid=0;
    while(low<=high){
        mid=(low+high)/2;
        if(a[mid]==Tofind) return mid;
        else if(a[mid]>Tofind) high=mid-1;
        else low=mid+1;
    }
    return -1;
}
int main(){
    srand((unsigned) time(NULL));
    int mode=0,n=0,a[25]={0},book=0;//book标记是否生成过数据
    while(1){//将程序保持在循环状态
        printmenu();
        scanf("%d",&mode);
        switch(mode){
            case 1:{
                n=input(a,20);
                if(n==-1) break;
                book=1;
                break;
            }
            case 2:{
                if(book==0){
```

```
            puts("还没有生成/输入过数据！");
            system("pause");
            break;
        }
        sort(a,n);
        puts("排序后的数组元素如下：");
        printArray(a,n);
        break;
    }
    case 3:{
        if(book==0){
            puts("还没有生成/输入过数据！");
            system("pause");
            break;
        }
        puts("请输入要查找的数据：");
        int Tofind=0,k=0;
        scanf("%d",&Tofind);
        k=Search_binary(a,n,Tofind);
        if(k==-1)
            printf("%d不是数组的一个元素！\n",Tofind);
        else
            printf("%d是数组中的第%d个元素。\n",Tofind,k+1);
        system("pause");
        break;
    }
    case 4:{
        puts("欢迎下次使用！");
        break;
    }
    default:{
        puts("无效操作");
        system("pause");
        break;
    }
```

```
        }
        if(mode==4) break;
    }
    return 0;
}
```

作业 12

P234 T1

若有定义和初始化如下：

```
int i,*p=&i;
```

则以下哪些表达式可以表示 i？哪些表达式可以表示 i 的地址？

(1) *p (2) &p (3) **p (4) *&p (5) &*p (6) &&p

(7) *i (8) &i (9) **i (10) *&i (11) &*i (12) &&i

作业反馈：很多同学漏了 (4) 也可以表示 i 的地址，其他基本没错。

参考答案：

(1) *p, (10) *&i 都能表示 i.

(4) *&p, (5) &*p ,(8) &i 都能表示 i 的地址。

仅有 (2) &p 能表示 p 的地址。

剩下 6 个表达式存在语法错误，无法通过编译。

P234 T5

若有定义和初始化如下：

```
int a[ ]=23,74,88,19,6,35,41,93,24,100;
int *p=a,*q=&a[4];
```

则以下表达式的值分别是多少？

(1) *p (2) p[2] (3) *(p+5) (4)*p+5 (5) q-p (6) *p-*q

(7) q[0] (8) q[2] (9) *(q-2) (10)q[-2] (11) p<q (12) *p<*q

作业反馈：这道题大家基本都没出错，少数同学对 (10)q[-2] 是否符合语法有所质疑，其实只要到编译器里试一下就知道这个表达方式没有语法问题，可以正常运行。

参考答案：

(1) 23 (2) 88 (3) 35 (4) 28 (5) 4 (6) 17

(7) 6 (8) 41 (9) 88 (10) 88 (11) 1 (12) 0

P234 T6

阅读如下程序，写出运行结果。

```
#include<stdio.h>
int main(){
    int a[]={0,1,2,3,4,5,6,7,8,9};
    int *p=a,*q=&a[9];
    printf("%d\n",*p++);
    printf("%d\n",*++p);
    printf("%d\n",*q--);
    printf("%d\n",*--q);
    while(p<=q)
        printf("%d %d\n",*p++,*q--);
    return 0;
}
```

作业反馈：基本没有同学出错。

参考答案：

```
0
2
9
7
2 7
3 6
4 5
```

note: 有同学疑惑`for(int i=0;i<n;i++)`与`for(int i=0;i<n;++i)`有什么区别，这里解释一下。从效果上来说，二者没有任何区别。在运行时间上，由于"i++"的实现方式是先创造一个临时变量存储 i+1 的值，在 i 的值使用完毕之后（当然在以上的语句里不需要使用 i 的值）进行赋值；而 ++i 是直接进行赋值 i=i+1 后再使用 i 的值，因此 i++ 比 ++i 多一步赋值操作，从而运行速度慢于 ++i.

对于一般的程序，即使将 i++ 全部换成 ++i，程序运行时间的减少也不如程序运行时间本身的波动幅度大。这种几乎无效的“优化”就是上面提到过的“卡常”的典型例子，所以说不需要去了解。