第五次实验

实验指导书P44

T3：

编写程序，猜一个在给定整数范围内产生的随机数，可以猜多次直到猜中。猜中后发一随机金额的红包。或者输入一个非数字退出程序。

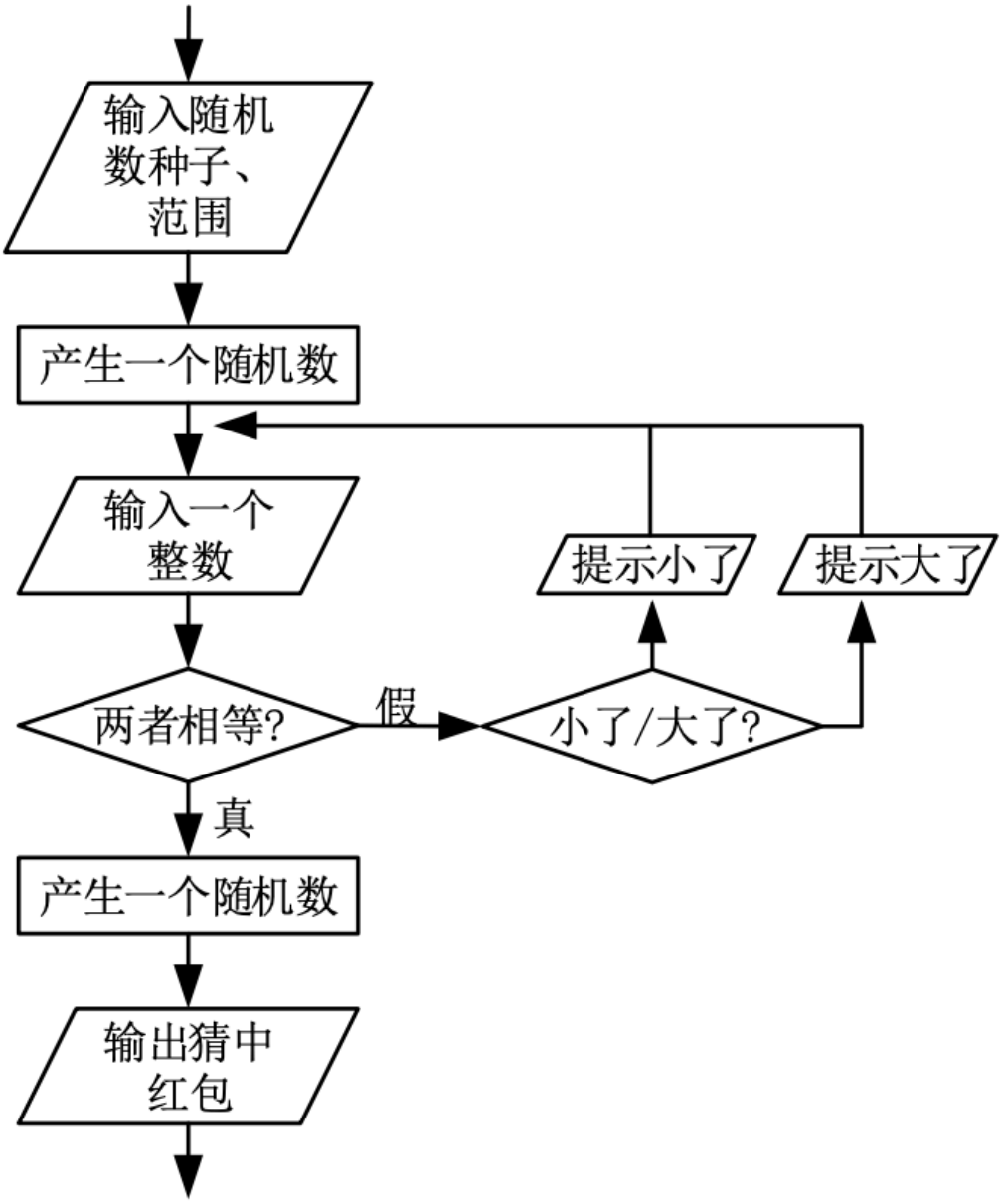
随机数是指随机发生的，不确定的，很难预测的数。随机数有真随机和伪随机两种，如掷骰子的点数、抛硬币的正反面等为真随机，而计算机软件计算或模拟的随机数则是伪随机数。真随机数与伪随机数的主要区别是真随机数是完全不可预知的，无法再现的;而伪随机 数因其算法或过程是确定的，故而可以预知或再现。

计算机中产生伪随机数的计算方法很多，比如 C 语言中的 rand 函数产生伪随机数的计算方法:

```
next=next*1103515245+12345;
rand_num=(unsigned)(next/65536)%32768;
```

此算法 执行一次就产生一个位于 0~32767 之间的随机整数。另外还有:

```
b=b*a-(long)(b*a);
f=fabs((float)(1/f)-(double)(1/f)*1e15);
```



程序设计要求:

> (1)按图 2.3 所示的流程图设计程序。
>
> (2)产生随机数的范围可以通过乘除与求余等运算进行限定。
>
> (3)用两种不同的产生随机数方法，分别产生要猜的数字和红包金额。要猜的随机数字的产生由键盘输入其随机数种子;红包金额的随机产生则利用之前的随机数作为种子。
>
> (4)程序运行时，利用折半的方法输入要猜的数，当与产生的随机数进行比较时相等即为猜中。

程序运行示例:

> 输入随机数种子:6
>
> 输入产生随机数的最小值:1
>
> 输入产生随机数的最大值:10
>
> 已在 1-10 范围内产生了一个随机数,请猜此数:6

> 你猜的数小了，请继续在 1-10 范围内猜数字:8
>
> 你猜的数小了，请继续在 1-10 范围内猜数字:9
>
> 你猜的数小了，请继续在 1-10 范围内猜数字:10
>
> 恭喜你猜中了!并获得 16 元的红包。

T4:背包问题的贪心法求解

有一个背包最大可以容纳 x 重量的物品，现在有 n 个物品，均有各自的重量和价值。问如何将这些物品装入这个背包里，使得背包里物品的价值最大，且其重量不能超过背包的最 大容量。
问题说明:

> 1)0-1 背包问题:每种物品仅 1 个，不可拆分，只能全部装入或不装入;
>
> 2)部分背包问题:每种物品仅 1 个，可以拆分，可以全部装入或按重量(或按比 例)拆分后装入。

编程要求:

> 1)用贪心法分别按价值和单位重量价值(物品的价值/物品的重量)求解 0-1 背 包问题，不需要得到最优解;
>
> 2)用贪心法按单位重量价值求解部分背包问题。
>
> 3)输入 n 个(3≤ n ≤10)物品的编号、重量和价值、以及背包可以容纳的重量 x，可以不按顺序输入，但程序里要根据需要进行排序。
>
> 4)打印输出背包里装入物品的总价值，和装了哪些物品以及其重量和价值 编程提示:1)可以用多个数组或结构体数组表示物品的属性(编号、重量、价值和单位重 量价值);
>
> 2)数组排序可每次从原数组里找最大或最小数存放到排序后的数组里。如:

```
#include <stdio.h>
int main()
{   float a[10],b[10];//将 a 数组从大到小排序后存入数组 b
    int a_flag[10]={0};//数组 a 中的数排序标志：0-没有排序，1-排序
    int i,j,k;//循环变量

    printf("输入 10 个浮点数:");
    for(i=0;i<10;i++)
        scanf("%f",&a[i]);

    for(i=0;i<10;i++)//数组 b 从大到小存储依次从数组 a 里找到的最大数
    {
        for(j=0;j<10;j++)
            if(a_flag[j]==0)
            {
                k=j;//先从没有排序的数中找一个假设最大的
                break;
            }
        for(j=0;j<10;j++)//搜索数组 a
        {
            if(a_flag[j]==0 && a[k]<a[j])//在剩下的数里找最大的
            {
                k=j;//记录没有排序的数里最大的
            }
        }
        b[i]=a[k];//按序存放到数组 b 中
        a_flag[k]=1;//标志数组 a[k]已经排序
    }

    printf("排序后：");

     for(i=0;i<10;i++)//打印排序后的数组
     {
         printf("%f ",b[i]);
     }
     return 0;
 }
```

程序运行示例:

> 输入背包的容量:5
>
> 输入物品的数量:3

> 输入 3 个物品的编号:1 2 3
>
> 输入 3 个物品的重量:1 2 3
>
> 输入 3 个物品的价值:60 100 120
>
> 0-1 背包问题的按价值贪心法求解:
>
> 3 号物品 重量:3.000000 价值:120.000000 装入背包
>
> 1 号物品 重量:2.000000 价值:100.000000 装入背包
>
> 装入背包的物品总价值为:220.000000
>
> 0-1 背包问题的按单位价值贪心法求解:
>
> 1 号物品 重量:1.000000 价值:60.000000,单位价值:60.000000,装入背包
>
> 2 号物品 重量:2.000000 价值:100.000000,单位价值:50.000000,装入背包
>
> 装入背包的物品总价值为:160.000000
>
> 部分背包问题的按单位价值贪心法求解:
> 1 号物品 重量:1.000000 价值:60.000000,单位价值:60.000000,装入背包
>
> 2 号物品 重量:2.000000 价值:100.000000,单位价值:50.000000,装入背包
>
> 3 号物品 重量:3.000000 价值:120.000000,单位价值:40.000000,装入背包
>
> 装入背包的物品总价值为:240.000000

T5:找零问题

现存的流通人民币面额有 100 元、50 元、20 元、10 元、5 元、1 元、5 角和 1 角等几种，现拿 100 元去购买物品，一共花了 x 元，请设计程序给出找零的方案，使之每次找零的张数最少。

编程要求:

> (1)输入购买物品的费用 x，并判断其合理性
>
> (2)对输入合理的费用，给出找零的方案(即每种钱币的数量)，使之找零的钱币张数最少
>
> (3)不足 1 角按照四舍五入进行找零

程序运行示例:

> 输入:请输入花费(单位:元):12.34
>
> 输出:找零87.7元，方案:50.0元1张 20.0元1张 10.0元1张 5.0元1张 1.0元2张 0.5元1张 0.1元2张

附加题：

T1:关于溢出

观察如下代码：

```
#include <stdio.h>
int main(){
    char c[16];
    char b[16];
    char a[16];
    gets(a);//或者scanf("%s",a)，但是其中不能带空格
    gets(b);
    gets(c);
    printf("%s\n",a);
}
```

输入以下三行数据，观察输出，并考虑为什么输出是这样。

```
Experience is th
e mother of wisd
om.
```

根据上面的原理，给下面的程序输入一行数据，使得程序输出 `Test OK.`

```
#include <stdio.h>
#include <string.h>
int main(){
    char b[16];
    char a[16];
    gets(a);
    if(!strcmp(b,"What's up?")) printf("Test OK.\n");
    else printf("Think more.");
    return 0;
}
```

T2

观察上面T4的背包问题，思考更快速的解法（只需回答问题，无需编程实现）

我们假设有n件物品，背包容量为m，第i件物品重量为w[i]，价值为v[i]，我们设数组：

$$f[n][m]，其中f[i][j]表示只考虑前i件物品，且最多占用j重量时的最优解。$$

请写出fij关于以上几个量的递推式。

提示，考虑决策：是否要放入第i件物品？如果要放，那是否要预留重量？如果不放，那么结果怎么计算？

注意，计算fij时，f(i-a)(j-b)都是可以使用的（a、b>=0）。

这个方法称为**动态规划**，01背包问题是动态规划最经典的问题。他的思想是将大规模问题的答案通过几个小规模问题的答案的最优解计算出来。

一般有这么几个步骤：

> （1）规划状态，比如定义上面的f数组
>
> （2）考虑状态的决策关系，写出状态转移方程（递推式）
>
> （3）根据方程的依赖关系，编程实现

如果有了上面的数组的递推式子和初始值，那么计算出fnm就是问题的解。

T3.编程题：求总和最大的连续子串

对于序列a，我们需要求出它的一个**连续**子串，使得这个子串的和最大。

比如：

1 -3 2 7 -8 9 -6 3

那么答案就是"2 7 -8 9"组成的子串。

要求：一次循环，一重。

提示：子串的和其实就是求区间的和，考虑作业中前缀和（前n项和）的做法，将区间和转换为端点相减。