

HUNAN UNIVERSITY

程序设计训练

报 告

学生姓名 杨廷元

学生学号 201926010513

专业班级 软件1905

指导老师 洪跃山

助 教 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_谭震\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2020 年 9 月 1 日

目 录

[题1：最少钱币数 3](#_Toc517944648)

[一、 问题描述 3](#_Toc517944649)

[二、 问题分析及边界条件 4](#_Toc517944650)

[四、 详细设计（从算法到程序） 4](#_Toc517944652)

[五、 样例设计与测试 4](#_Toc517944653)

[六、 分析与总结 6](#_Toc517944654)

[题2：数塔 7](#_Toc517944648)

[一、 问题描述 7](#_Toc517944649)

[二、 问题分析及边界条件 7](#_Toc517944650)

[三、 算法设计 8](#_Toc517944651)

[四、 详细设计（从算法到程序） 8](#_Toc517944652)

[五、 样例设计与测试 8](#_Toc517944653)

[六、 分析与总结 10](#_Toc517944654)

[题3：蛇形矩阵 11](#_Toc517944648)

[一、 问题描述 11](#_Toc517944649)

[二、 问题分析及边界条件 11](#_Toc517944650)

[三、 算法设计 11](#_Toc517944651)

[四、 详细设计（从算法到程序） 12](#_Toc517944652)

[五、 样例设计与测试 12](#_Toc517944653)

[六、 分析与总结 13](#_Toc517944654)

[题4：部分A+B 14](#_Toc517944648)

[一、 问题描述 14](#_Toc517944649)

[二、 问题分析及边界条件 14](#_Toc517944650)

[三、 算法设计 14](#_Toc517944651)

[四、 详细设计（从算法到程序） 15](#_Toc517944652)

[五、 样例设计与测试 15](#_Toc517944653)

[六、 分析与总结 16](#_Toc517944654)

[题5：个位数字统计 17](#_Toc517944648)

[一、 问题描述 17](#_Toc517944649)

[二、 问题分析及边界条件 17](#_Toc517944650)

[三、 算法设计 17](#_Toc517944651)

[四、 详细设计（从算法到程序） 18](#_Toc517944652)

[五、 样例设计与测试 18](#_Toc517944653)

[六、 分析与总结 19](#_Toc517944654)

# 题1：最少钱币数

## 问题描述

【问题描述】这是一个古老而又经典的问题。用给定的几种钱币凑成某个钱数，一般而言有多种方式。例如：给定了 6 种钱币面值为 2、5、10、20、50、100，用来凑 15 元，可以用 5 个 2 元、1个 5 元，或者 3 个 5 元，或者 1 个 5 元、1个 10 元，等等。显然，最少需要 2 个钱币才能凑成 15 元。  
        你的任务就是，给定若干个互不相同的钱币面值，编程计算，最少需要多少个钱币才能凑成某个给出的钱数。

【输入形式】输入可以有多个测试用例。每个测试用例的第一行是待凑的钱数值 M（1 <= M<= 2000，整数），接着的一行中，第一个整数 K（1 <= K <= 10）表示币种个数，随后是 K个互不相同的钱币面值 Ki(1 <= Ki <= 1000)。输入 M=0 时结束。

【输出形式】每个测试用例输出一行，即凑成钱数值 M 最少需要的钱币个数。如果凑钱失败，输出“Impossible”。你可以假设，每种待凑钱币的数量是无限多的。

## 问题分析及边界条件

## 从规模最小的m开始。当m=0时，即我们需要多少个币来凑够0元呢？ 由于1，3，5，7都大于0，即没有比0小的币值，因此凑够0元我们最少需要0个币。 (em......Interesting，这个分析很傻是不是？别着急，这个思路有利于我们理清动态规划究竟在做些什么。) 为了方便我们用dp[m]=c来表示凑够m元最少需要c个硬币。于是我们就得到了dp[0]=0， 表示凑够0元最小需要0个硬币。当m=1时，只有面值为1元的硬币可用， 因此我们拿起一个面值为1的硬币，接下来只需要凑够0元即可，而这个是已经知道的dp[0]=0。所以，dp[1]=dp[1-1]+1=dp[0]+1=0+1=1。当m=2时， 仍然只有面值为1的硬币可用，于是我拿起一个面值为1的硬币， 接下来我只需要再凑够2-1=1元即可(记得要用最小的硬币数量)，而这个答案也已经知道了。 所以dp[2]=dp[2-1]+1=dp[1]+1=1+1=2。分析到这里，聪明的你可能已经看出端倪，没看出来没关系，接下来让我们看看m=3时的情况。当m=3时我们能用的硬币就有两种了：1元的和3元的( 5元的仍然没用，因为你需要凑的数目是3元，5元面值太大了)。 既然能用的硬币有两种，我就有两种方案。如果我拿了一个1元的硬币，我的目标就变为了： 凑够3-1=2元需要的最少硬币数量。即dp[3]=dp[3-1]+1=dp[2]+1=2+1=3。 这个方案说的是，我拿3个1元的硬币；第二种方案是我拿起一个3元的硬币， 我的目标就变成：凑够3-3=0元需要的最少张纸币。即dp[3]=dp[3-3]+1=dp[0]+1=0+1=1. 这个方案说的是，我拿1个3元的硬币。好了，这两种方案哪种更优呢？ 记得我们的问题是要用最少的硬币数量来凑够3元。所以， 选择dp[3]=1，怎么来的呢？具体是这样得到的：dp[3]=min(dp[3-1]+1, dp[3-3]+1)。

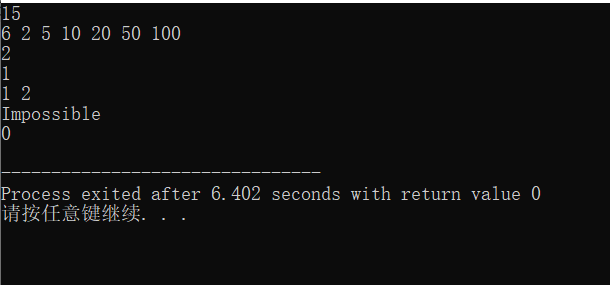
## 三、算法设计

## dp[m]表示凑够m元需要的最少硬币数量，我们将它定义为该问题的"状态"， 这个状态是怎么找出来的呢？是根据子问题定义状态。你找到子问题，状态也就浮出水面了。 最终我们要求解的问题，可以用这个状态来表示：dp[8]，即凑够8元最少需要多少个硬币。 那状态转移方程是什么呢？既然我们用dp[m]表示状态，那么状态转移方程自然包含dp[m]， 上文中包含状态 dp[m] 的方程是：dp[3]=min(dp[3-1]+1, dp[3-3]+1)。没错， 它就是状态转移方程，描述状态之间是如何转移的。当然，我们要对它抽象一下，dp[m] = min(dp[ m-v[i] ]+1,dp[m])，其中 m-v[i] >=0，v[i] 表示第i个硬币的面值，方程的含义是拿出一个面值为 v[i] 的硬币后，凑够 m-v[i] 元至少需要的硬币数目（dp[m-v[i] ]）+1和凑够m元至少需要的硬币数目（dp[m]）相比较，取较小的存入dp[m]。

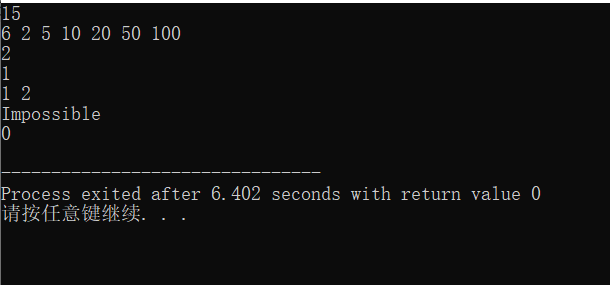
## 四、详细设计（从算法到程序）

## 五、样例设计与测试

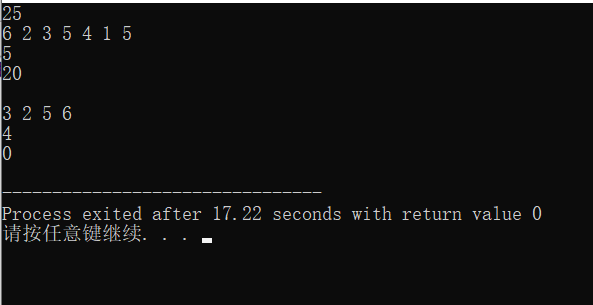
样例输入1：



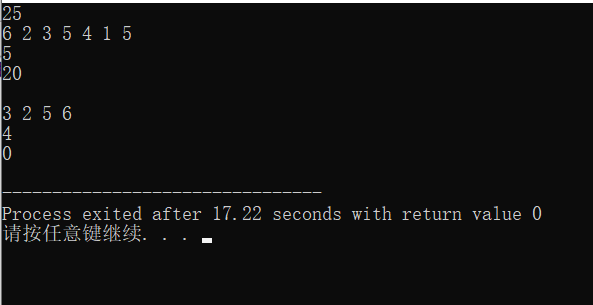
样例输出1：



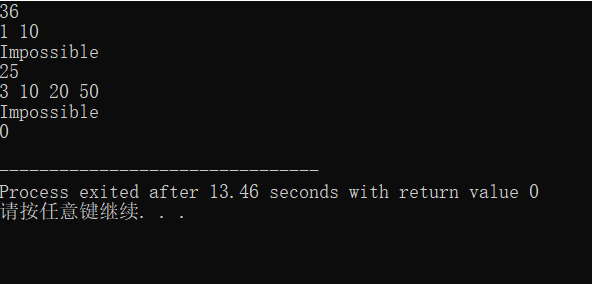
样例输入2：



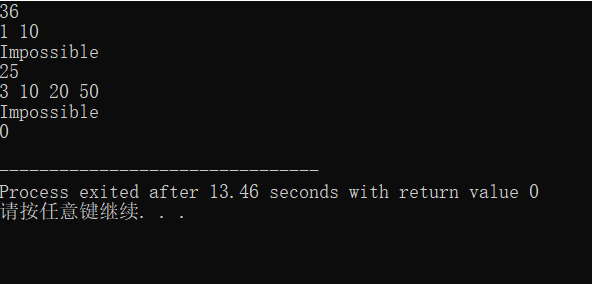
样例输出2：



样例输入3：



样例输出3：



## 六、分析与总结

1， 在写程序的时候发现， 虽然我学过一点STL,但在用起来的时候非常的不熟练，需要边查资料边写，以前大多用c语言写程序，现在有必要多写一点c++程序，多用STL，提高编程能力。

2，这个程序的查找还可以优化一下，可以用二分法查找，找到小于或者剩余的钱数。

3，题目要求的输入限制没有检测。

# 题2：数塔

## 问题描述

【问题描述】

给定一个数塔，如下图所示。在此数塔中，从顶部出发，在每一节点可以选择走左下或右下，一直走到底层。请找出一条路径，使路径上的数值和最大。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | 9 |  |  |  |  |
|  |  |  | 12 |  | 15 |  |  |  |
|  |  | 10 |  | 6 |  | 8 |  |  |
|  | 2 |  | 18 |  | 9 |  | 5 |  |
| 19 |  | 7 |  | 10 |  | 4 |  | 16 |

【输入形式】

输入时第一行一个整数n，表示该数塔的行数，其余n行表示该塔每行的数值

【输出形式】

输出包含两行，第一行为最大路径上的数值之和， 第二行n个数字为从上而下最大路径数值

## 问题分析及边界条件

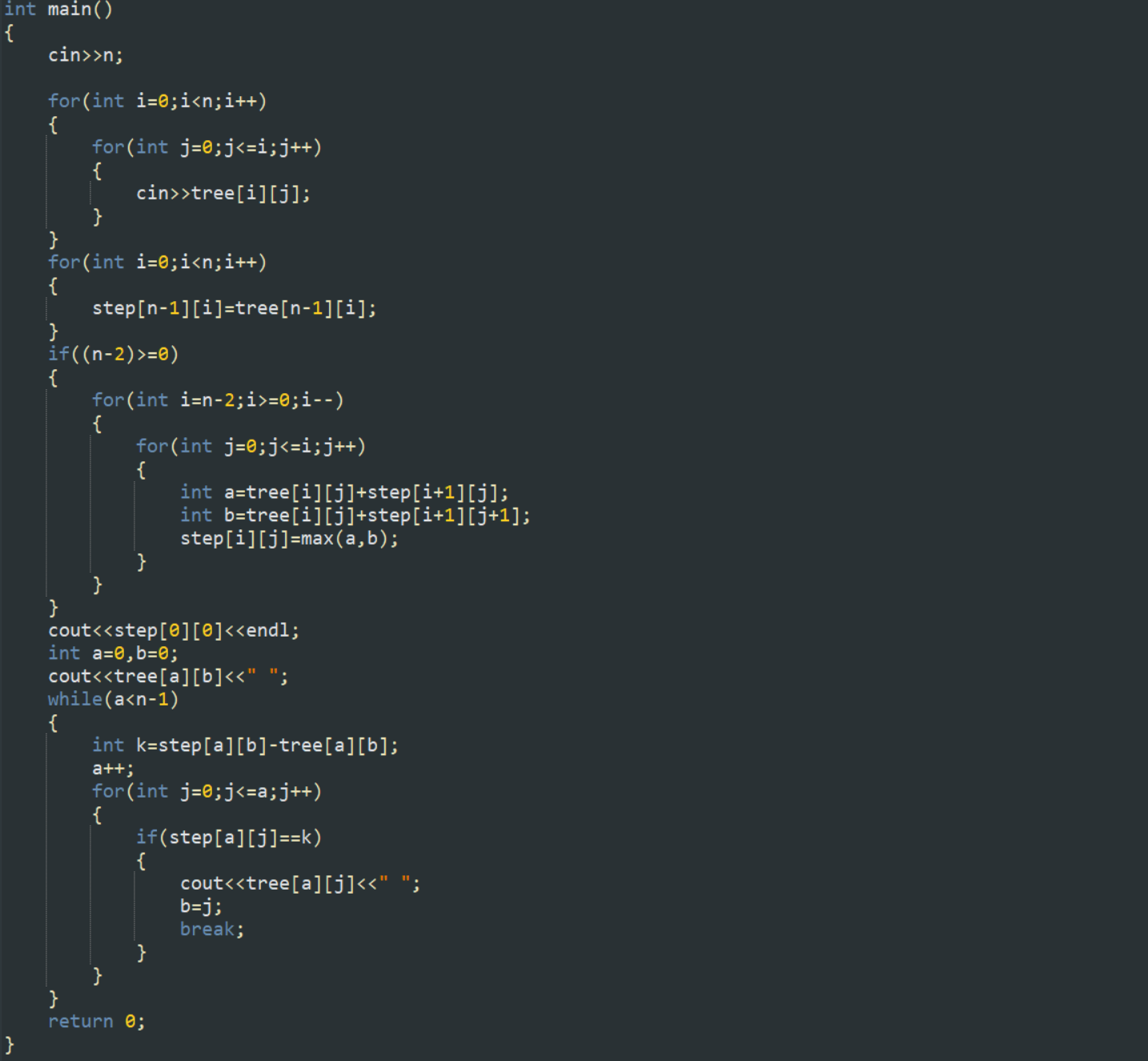
## 在用动态规划考虑数塔问题时可以从下往上计算。 从顶点出发时选择向左走还是向右走取决于是向左走能取到最大值还是向右走能取到最大值，所以在选择之前必须要先求出左右两条路径的值。同理下一层再往下寻找道路时也要线求出左右两条路径的值。以此类推目前一直到第四层中的节点向第五层中选择。 以一层数据的选择为一个阶段，那么这个问题的求解可以分为4个阶段。 第一阶段：从第四层数据向下选择 第二阶段：从第三层数据向下选择 第三阶段：从第二层数据向下选择 第四阶段：从第一层数据向下选择

## 三、算法设计

## 第一阶段过程分析： 所以第一步对第五层的8个数据，做如下四次选择： 如果经过2，则在第五层的19和7中选择19； 如果经过18，则在第五层的7和10中选择10； 如果经过9，则在第五层的10和4中选择10； 如果经过5，则在第五层的4和16中选择16；

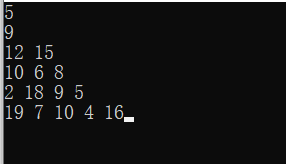
经过一次决策，问题降了一阶。5层数塔问题转换成4层数塔问题，此时可以另外创建一个数组temp来存放选择之后的数塔，推到最后的temp数组。

## 四、详细设计（从算法到程序）

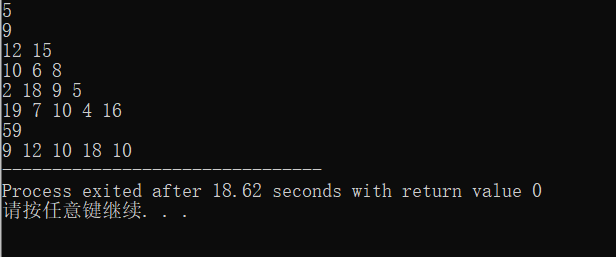


## 五、样例设计与测试

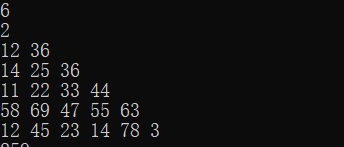
样例输入1：



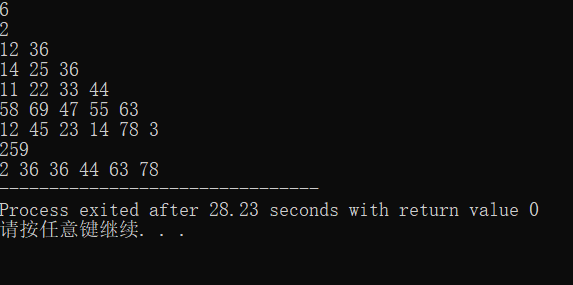
样例输出1：



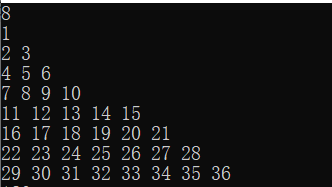
样例输入2：



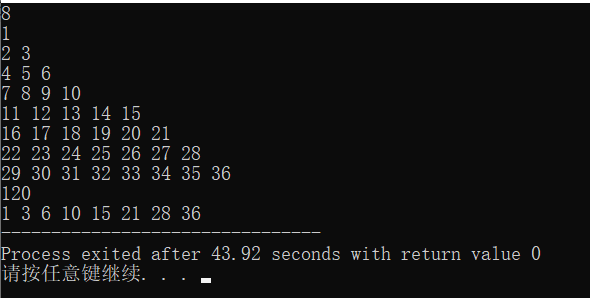
样例输出2



样例输入3：



样例输出3：



## 六、分析与总结

1．对问题的分析更加深入，从算法到代码的实现变得容易了。

2，通过不断的测试和调试，最终得到了最优代码，但算法有待提高。

# 题3：蛇形矩阵

## 问题描述

【问题描述】蛇形矩阵是由 1 开始的自然数依次排列成的一个矩阵上三角形

【输入形式】 正整数 N表示层数，N 不大于 100

【输出形式】输出一个 N 行的蛇形矩阵，矩阵三角中同一行的数字用一个空格分开，行尾不要多余的空格。

## 问题分析及边界条件

## 要实现以上程序需要定义int数据类型以及整型数组，用for循环和递归算法来实现蛇形矩阵，并且只用一个主函数来实现；

## 三、算法设计

1.输入数据类型与主函数内数据定义与初始化

intn,i,j,k;

inta[N][N];

2. 输入语句与输出语句的实现

Cin>>n

Cout<<a[i][j];

3.for循环的实现

 for(i=0;i<n;i++)

  for(j=0;j<n-i;j++)

{

        k=i+j+1;

      a[i][j]=k\*(k-1)/2+j+1;

}

 for(i=0;i<n;i++)

 {

  for(j=0;j<n-i;j++)

  {

  Cout<<a[i][j]；

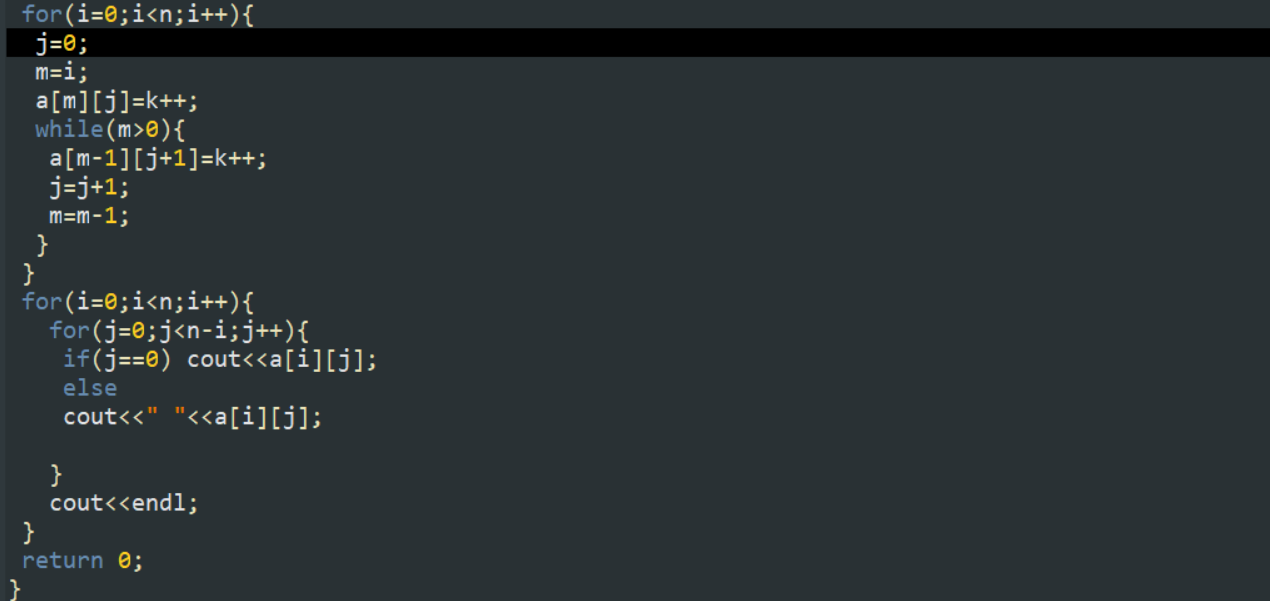
  }

  Cout<<endl；

}

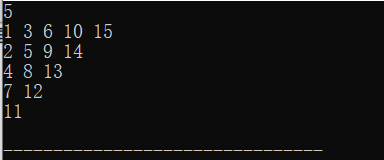
。

## 四、详细设计（从算法到程序）

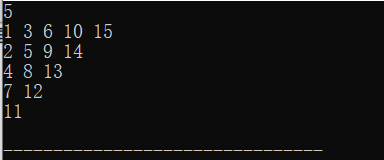


## 五、样例设计与测试

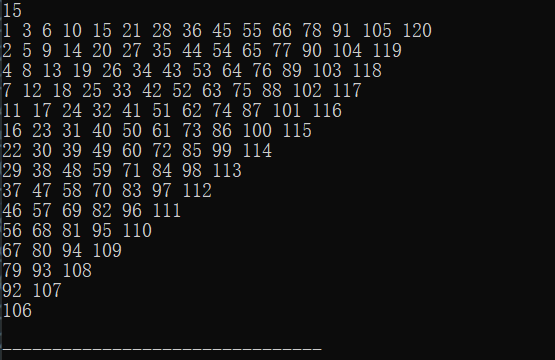
样例输入1：



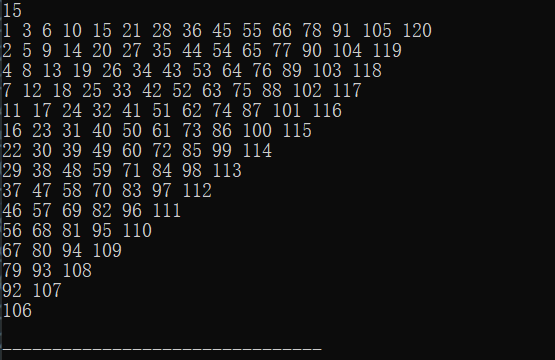
样例输出1：



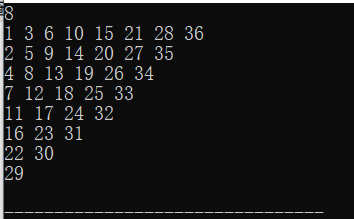
样例输入2：



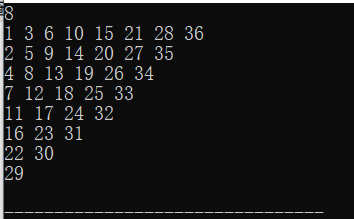
样例输出2



样例输入3：



样例输出3：



## 六、分析与总结

1．程序的输入输出运用的都是整型，并且用for循环和数组来实现，当输入n(n大于0)的数程序开始执行1号for循环，知道i=n结束1号for循环。然后执行2号for循环，逐一把数打印出来当i=n时结束程序。

2，通过不断的测试和调试，最终得到了最优代码，但算法有待提高。

# 题4：部分A+B

## 一、问题描述

【问题描述】

正整数A的“DA（为1位整数）部分”定义为由A中所有DA组成的新整数PA。例如：给定A = 3862767，DA = 6，则A的“6部分”PA是66，因为A中有2个6；给定A = 3862767，DA = 1，则A的“1部分”PA是0，因为A中有0个1。

现给定A、DA、B、DB，请编写程序计算PA + PB。

【输入形式】

输入在一行中依次给出A、DA、B、DB，中间以空格分隔其中0 < A, B < 1010。

【输出形式】

在一行中输出PA + PB的值。

## 二、问题分析及边界条件

## 只用主要数的大小，要求是小于10^10。所以int是不行的，int是4字节，可以用八字节的长整型long long来存储最终答案。另外要注意，如果没有对应的数字则PX=0P\_{X}=0PX​=0

## 三、算法设计

分别找到A和B中PA和PB，利用关系式计算值

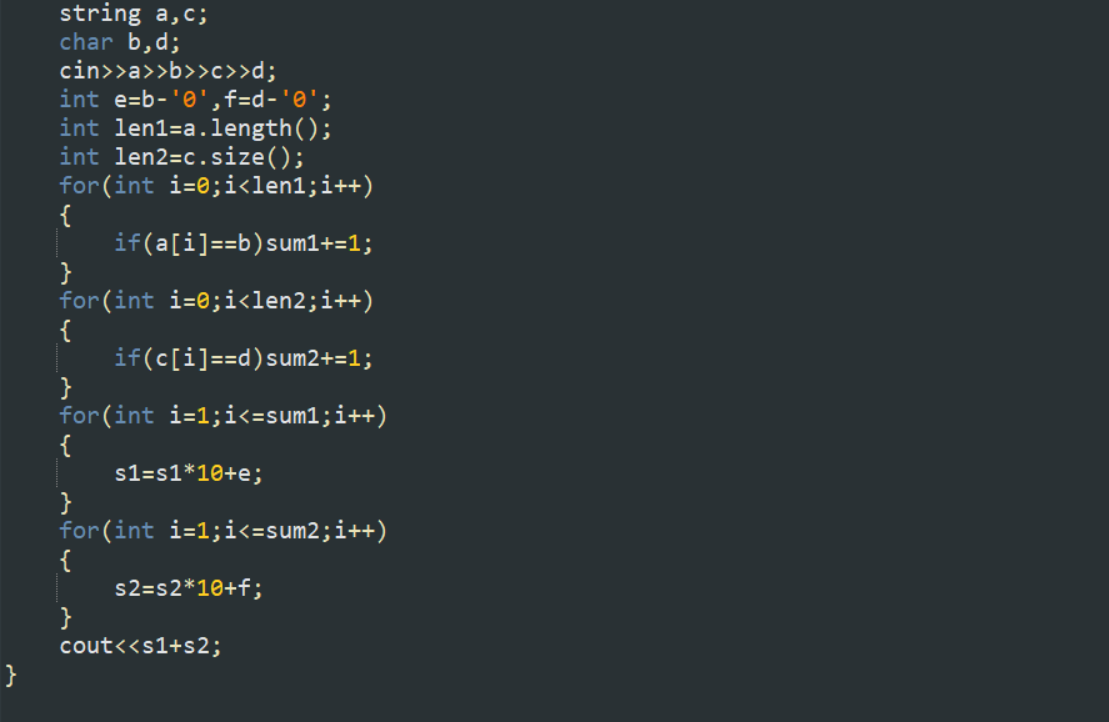
针对1:可以将A储存到字符串或者字符数组中，遍历查找整型PA或PB-

针对2:找到关系式a= a \* 10+tempA'

tempA:初始输入的PAa:最终计算的值

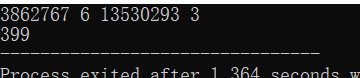
代码就是将字符串换成字符数组，迭代器遍历换成下标遍历。

## 四、详细设计（从算法到程序）

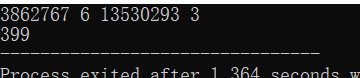


## 五、样例设计与测试

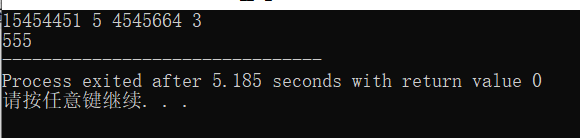
样例输入1：



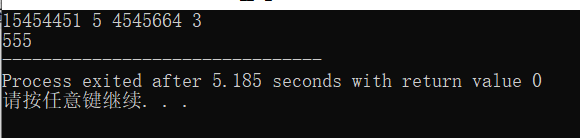
样例输出1：



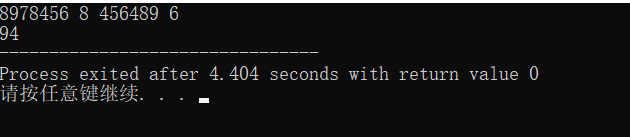
样例输入2：



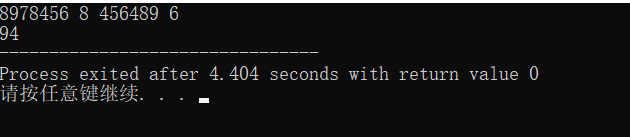
样例输出2



样例输入3：



样例输出3：



## 六、分析与总结

1．通过对问题的分析找到问题的核心，加之以代码实现，更能轻松解决问题

2，分析问题的能力有待提高，算法方面需进一步精进