



HUNAN UNIVERSITY

程序设计训练

报 告

2023 年 7 月 29 日

目 录

[题1：错误的里程表(ID: 16369) 4](#_Toc517944648)

[一、 问题描述 4](#_Toc517944649)

[二、 问题分析及边界条件 5](#_Toc517944650)

[三、 算法设计 5](#_Toc517944651)

[四、 详细设计（从算法到程序） 5](#_Toc517944652)

[五、 样例设计与测试 6](#_Toc517944653)

[六、 分析与总结 7](#_Toc517944654)

[题2：内存管理(ID: 16368) 8](#_Toc517944648)

[一、 问题描述 8](#_Toc517944649)

[二、 问题分析及边界条件 9](#_Toc517944650)

[三、 算法设计 1](#_Toc517944651)0

[四、 详细设计（从算法到程序） 1](#_Toc517944652)0

[五、 样例设计与测试 1](#_Toc517944653)2

[六、 分析与总结 1](#_Toc517944654)4

[题3：买房与选房(ID: 16375) 1](#_Toc517944648)4

[一、 问题描述 1](#_Toc517944649)4

[二、 问题分析及边界条件 1](#_Toc517944650)5

[三、 算法设计 1](#_Toc517944651)5

[四、 详细设计（从算法到程序） 1](#_Toc517944652)6

[五、 样例设计与测试 2](#_Toc517944653)0

[六、 分析与总结 2](#_Toc517944654)0

[题4：缺席考试的是谁？(ID: 16391) 2](#_Toc517944648)1

[一、 问题描述 2](#_Toc517944649)1

[二、 问题分析及边界条件 2](#_Toc517944650)2

[三、 算法设计 2](#_Toc517944651)2

[四、 详细设计（从算法到程序） 2](#_Toc517944652)3

[五、 样例设计与测试 24](#_Toc517944653)

[六、 分析与总结 2](#_Toc517944654)6

[题5：数圈(ID: 16392) 2](#_Toc517944648)6

[一、 问题描述 2](#_Toc517944649)6

[二、 问题分析及边界条件 2](#_Toc517944650)7

[三、 算法设计 2](#_Toc517944651)7

[四、 详细设计（从算法到程序） 2](#_Toc517944652)8

[五、 样例设计与测试 2](#_Toc517944653)9

[六、 分析与总结 3](#_Toc517944654)0

[题6：最大报销额(ID: 16414) 3](#_Toc517944648)0

[一、 问题描述 3](#_Toc517944649)0

[二、 问题分析及边界条件 3](#_Toc517944650)1

[三、 算法设计 3](#_Toc517944651)1

[四、 详细设计（从算法到程序） 3](#_Toc517944652)2

[五、 样例设计与测试 3](#_Toc517944653)3

[六、 分析与总结 3](#_Toc517944654)5

[题7：公交系统(ID: 16402) 3](#_Toc517944648)5

[一、 问题描述 3](#_Toc517944649)5

[二、 问题分析及边界条件 3](#_Toc517944650)7

[三、 算法设计 3](#_Toc517944651)7

[四、 详细设计（从算法到程序） 3](#_Toc517944652)8

[五、 样例设计与测试 3](#_Toc517944653)9

[六、 分析与总结 4](#_Toc517944654)0

[题8：Web导航(ID: 16419) 4](#_Toc517944648)0

[一、 问题描述 4](#_Toc517944649)0

[二、 问题分析及边界条件 4](#_Toc517944650)2

[三、 算法设计 4](#_Toc517944651)2

[四、 详细设计（从算法到程序） 4](#_Toc517944652)2

[五、 样例设计与测试 4](#_Toc517944653)3

[六、 分析与总结 4](#_Toc517944654)8

[题9：](#_Toc517944648)[相等的多项式](http://202.197.98.89/assignment/programList.jsp?proNum=8&assignID=949)[(ID: 16438) 4](#_Toc517944648)8

[一、 问题描述 4](#_Toc517944649)8

[二、 问题分析及边界条件 5](#_Toc517944650)0

[三、 算法设计 5](#_Toc517944651)0

[四、 详细设计（从算法到程序） 5](#_Toc517944652)1

[五、 样例设计与测试 5](#_Toc517944653)2

[六、 分析与总结 5](#_Toc517944654)4

[题10：疫情期间(ID:16430) 5](#_Toc517944648)4

[一、 问题描述 5](#_Toc517944649)4

[二、 问题分析及边界条件 5](#_Toc517944650)6

[三、 算法设计 5](#_Toc517944651)6

[四、 详细设计（从算法到程序） 5](#_Toc517944652)7

[五、 样例设计与测试 5](#_Toc517944653)8

[六、 分析与总结 5](#_Toc517944654)9

# 题1：错误的里程表(ID:16369)

## 问题描述

【问题描述】

三月八日，小明买了台新车。但很快小明发现汽车的里程表有问题：里程表上每一位都不显示数字3和数字8，也就是说直接从数字2跳到数字4，直接从数字7跳到数字9。

小明纳闷：这车到底行驶里程是多少。现在，小明向你求助：根据里程表显示的数字，给出真实的行驶里程。

【输入形式】

输入有多组测试数据。

输入第一行正整数T，表示有多少组测试数据。

后面有T行，每行一个非负整数，表示里程表显示数字，里面不含有数字3和8。该数字不超过10位。

40%的测试数据组数T  10≤T≤ 102；

30%的测试数据组数T  102≤T≤ 103；

20%的测试数据组数T  103≤T≤ 104；

10%的测试数据组数T  104≤T≤ 105；

【输出形式】

对于每组测试数据，输出一个整数占一行：真实的行程里程。

【样例输入】

6

0

1

12

159

111224459

124567976

【样例输出】

0

1

10

103

19212007

21913077

## 问题分析及边界条件

要处理的问题：

1. 根据给出的错误里程表输出正确的里程表，观察可知此问题是将八进制转化为十进制；

边界条件：

1. 错误里程表不显示数字3，8；
2. 数字个数不超过10；
3. 有多组数据

## 算法设计

*通过上述分析，将采用下列方法解决相关问题：*

1. *使用容器 / 数组保存每一位数字；*
2. *将错误里程表的数字修正为八进制；*
3. *将八进制转化为十进制；*
4. *输出转化后的十进制；*

## 详细设计（从算法到程序）

1. 用int型变量保存输入的数字；
2. 使用STL中vector容器保存每个位上的数字；
3. 遍历vector容器，将每个数字修正，即数字9改为7，数字4～7减1，其余数字不变；
4. 再遍历一次vecor容器，将修正后的八进制转化十进制，用int型变量保存；
5. 输出最终int型变量；

## 样例设计与测试

测试样例一：一组数据，10个数字

样例输入：

1

124974599

样例输出：22013759



测试样例二：正常输入

样例输入：

2

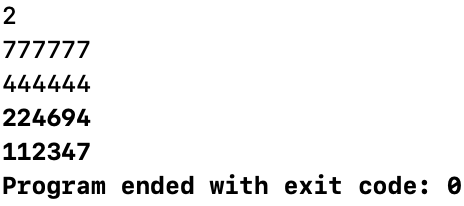
777777

444444

样例输出：

**224694**

**112347**



测试样例三：正常输入

样例输入：

3

761564719

4515

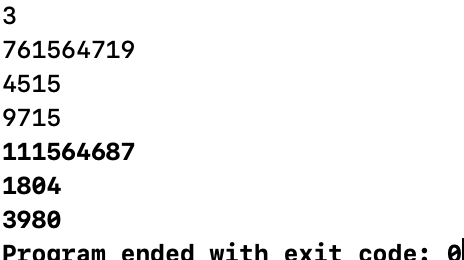
9715

样例输出：

**111564687**

**1804**

**3980**



## 分析与总结

1. 掌握进制间的转换；
2. 掌握vector容器的使用

# 题2：内存管理(ID:16368)

## 一、问题描述

【问题描述】

       离第一个操作系统HNU-OS发布已经没有多少时间了，但它的一些组件还没有完成，内存管理器就是其中之一。根据开发人员的计划，在第一个版本中，内存管理器将非常简单和直观。它将支持三个操作：

* alloc n —— 分配n个字节内存，返回已分配块的正整数标识符x(x初始值为0，每次分配增长1)
* erase x —— 删除标识符x所在的块
* defragment —— 整理空余内存碎片，将所有块尽量靠近内存的开始位置，并保持各自的顺序

       在此情况下，内存模型非常简单，它是一个m字节的序列，为了方便起见，从第一个字节到第m字节进行编号。

       第一个操作alloc n有一个参数n，表示被分配的内存块大小。在处理此操作时，内存中将分配n个连续字节的空闲块。 如果这些块的数量超过一个，则优先选择最接近内存开始(即第一个字节)的块。 所有这些字节都被标记为非空闲，内存管理器返回一个32位整数数字令牌，代表该块的标识符。 如果不可能分配这样大小的空闲块，则返回NULL。

       第二个操作erase x以x为参数，表示某个块的标识符。此操作释放系统内存，将此块的字节标记为空闲以供进一步使用。 如果此标识符没有指向先前分配的块(该块尚未被释放)，则返回ILLEGAL\_ERASE\_ARGUMENT。

       最后一个操作defragment没有任何参数，只会使占用的内存部分更接近内存的开始，而不会更改它们各自的顺序。

       在当前的实现中，将使用从1开始的连续整数作为标识符。每个成功的alloc操作过程都应该返回接下来的编号。不成功的alloc操作不影响计数。

       编写内存管理器的实现，为每个alloc命令输出返回的值，为所有失败的erase命令输出ILLEGAL\_ERASE\_ARGUMENT。

【输入形式】

       输入数据的第一行包含两个正整数t和m（1<=t<=500, 1<=m<=105)，其中t表示需要内存管理器来处理的操作个数，m表示有效的内存字节大小。接下来的t行每一行代表一个操作。  
【输出形式】

       输出有多行，每行或者是alloc操作的结果，或者是失败的erase操作的结果ILLEGAL\_ERASE\_ARGUMENT。其顺序与输入的操作次序一致。

【样例输入】

6 10  
alloc 5  
alloc 3  
erase 1  
alloc 6  
defragment  
alloc 6

【样例输出】

1  
2  
NULL  
3

## 二、问题分析及边界条件

要处理的问题：

1. 实现三个内存管理操作，输出相应结果；
2. 实现内存分配，记录已分配的内存块数；
3. 实现内存删除，删除对应内存块；
4. 实现内存整理，将内存块连续保存；

边界条件：

1. 只有一个内存，有多次操作，但不超过500次；
2. 题目中内存的大小不超过105；

## 三、算法设计

通过上述分析，将采用下列方法解决相关问题：

1. 使用整型vector容器，模拟内存（0表示此字节内存未被占用）；
2. 用循环实现对的内存多次操作；
3. 字符串保存操作指令；
4. 用整型变量保存内存大小以及已分配的内存块数；
5. 根据题目要求进行模拟，修改整型vector相应的元素值（分配时，将对应位置的元素置为当前内存块的序号，让删除操作更简单，删除时置为0）；
6. 每次操作后根据情况输出结果；

## 四、详细设计（从算法到程序）

## 1.全局变量声明：

**int** m;                  //内存大小

vector<**int**> memory;     //vecotr容器模拟内存

**int** stack = 0;              //已分配的内存块数

2.程序主框架设计：

**int** main(**int** argc, **const** **char** \* argv[]) {

**int** T; //操作次数

**……**         //输入操作次数、内存大小

    memory.resize(m, 0);    //内存初始化

**for** (**1 到 T**)

    {

        string s;

        cin >> s;           //输入操作指令

        //根据操作指令模拟操作

**if** (s == "alloc")

        {

**int** n; //要发分配的内存大小

            ……              //输入要发分配的内存大小

            alloc(n); //模拟分配操作

        }

**else** **if** (s == "erase")

        {

**int** pos; //要删除的内存块

            ……          //输入要删除的内存块

            myerase(pos); //模拟删除操作

        }

**else**

        {

            defragment(); //模拟整理操作

        }

    }

**return** 0;

}

1. 函数alloc用于模拟分配操作：

**void** alloc (**int** n)

{

**bool** flag = 0; //判断是否能分配所需大小的内存块

**for** (**int** i = 0; i < m; i++)

    {

**……**

//遍历内存找到第一个可以分配的位置去分配内存块，

//将这n个连续的元素值置为目前分配的内存块序号

**if** (flag == 1) //分配成功退出循环

**break**;

    }

**if** (flag == 1)

    {

        stack++; //内存块数+1

        cout << stack << "\n"; //能分配，输出已分配的内存块数

    }

**else**

        cout << "NULL" << "\n"; //不能分配输出NULL

}

1. myerase函数模拟删除操作：

**void** myerase (**int** pos)

{

**bool** iferase = 0; //判断操作是否成功

**for** (**int** i = 0; i < m; i++) //遍历内存寻找要删除的内存块

{

**if** (找到要删除的内存块)

        {

            …… //将对应位置元素置为0

            iferase = 1; //操作成功，iferase置为1

        }

}

**if** (不成功)

        cout << "ILLEGAL\_ERASE\_ARGUMENT" << "\n";

}

## 样例设计与测试

样例测试一：正常操作

样例输入：

6 6

alloc 1

alloc 2

alloc 3

erase 2

defragment

alloc 2

样例输出：

**1**

**2**

**3**

**4**

## 截屏2023-07-29 17.15.40

样例测试二：有失败的erase

样例输入：

6 6

alloc 1

alloc 2

alloc 3

erase 2

erase 2

erase 4

样例输出：

**1**

**2**

**3**

**ILLEGAL\_ERASE\_ARGUMENT**

**ILLEGAL\_ERASE\_ARGUMENT**

## 截屏2023-07-29 17.19.01

样例测试三：有失败的alloc

样例输入：6 10

alloc 3

alloc 5

alloc 2

alloc 2

erase 2

alloc 3

样例输出：

**1**

**2**

**3**

**NULL**

**4**

## 截屏2023-07-29 17.23.47

## 分析与总结

1. 掌握STL的vecotr容器的使用；

# 题1：买房与选房(ID: 16375)

## 一、问题描述

【问题描述】

      在 ****X****国许多一线城市住房非常紧张，政府部门制定了相关的政策，重点满足住房刚性需求（住房面积为0，社保缴纳必须超过2年），然后才能照顾改善性需求（住房面积大于0）。

   具体的原则为：

* 对于刚性需求，缴纳社保月数多者优先
* 对于改善性需求，现有自有住房面积小者优先

       由于房源有限，为公平起见，开发商在不违背上述原则下特意指定同等条件下申报时间同时作为排队的条件，时间越早优先级越高。

       最近有一批新楼盘准备开盘，总共有 ****m**** （≤1000）套房，所有的网上申报工作都已经完成并保存到二进制文件house.bin中，申请者提交了自己的基本材料，格式为：身份证号（18位，加1位空字符'\0'，共19位）、社保缴纳月数、自有住房面积、申报时间(格式为：**MM**-**DD**-**YYYY**，10位字符串，加1位空字符'\0'，共11位)，社保缴纳月数、自有住房面积均为整数，文件最后为总报名人数 ****n****（≤105）。

       申请者可以通过身份证号查询最终的结果。

【输入形式】

      输入的第一行为两个正整数 ****m****（≤1000）和 *****T*****（ ****T ≤ n****），分别表示本次开盘的楼盘可供申请的套数以及查询的组数

       接下来的 ****T****行，每行为一个18位的字符串，表示需要查询的身份证号  
【输出形式】

       输出为 ****T****行，对应每个查询的输出结果：

       1. 申请者不符合购房条件或排位超出了所推出的房源数量不能中签，则输出"Sorry";

       2. 申请者符合购房条件，且该名次人数为1人，则直接输出一个整数，表示选房顺序号;

       3. 申请者符合购房条件，且该名次人数有多人，同时人数不大于所剩房源数量，则直接输出用空格分隔的两个整数，表示选房顺序号区间;

       4. 申请者符合购房条件，且该名次人数有多人，同时人数大于所剩房源数量，则输出用/分隔两个整数，如 ****A****/****B****，表示 ****B****人中选 ****A**** 人，选房顺序为排名倒数 ****A**** 名范围。

【样例输入】

9 6

350102200609166049

350102200609163286

250342323545313434

130502201805070787

110101196003074525

430102201102181455

【样例输出】

2

3 4

Sorry

6

2/3

Sorry

## 二、问题分析及边界条件

要处理的问题：

1. 从二进制文件中读取数据；
2. 申请者买房的顺序按先比住房，再比社保缴纳月数，最后比申请时间的比较顺序进行排列；
3. 根据查询的申请者位次，判断是否有同位申请者，是否超出名额，输出查询的结果；

边界条件：

1.二进制文件数据格式为：身份证号（18位，加1位空字符'\0'，共19位）、社保缴纳月数、自有住房面积、申报时间(格式为：MM-DD-YYYY，10位字符串，加1位空字符'\0'，共11位)，社保缴纳月数、自有住房面积均为整数，文件最后为总报名人数 n（≤105）；

2.刚性需求的申请者的社保缴纳需超过2年；

## 三、算法设计

通过上述分析，将采用下列方法解决相关问题：

1. 使用结构体数组保存每个申请者的身份证号、住房面积、社保缴纳月数和申请日期；
2. 用文件指针读取二进制文件
3. 用插入排序按比较的优先级对申请者排序；
4. 用整型数组储存申请者的位次；
5. 根据输入的身份证号查询并输出结果；

## 四、详细设计（从算法到程序）

## 1.声明结构体

**struct** people

{

**char** id[19];                  /\* 身份证号码 \*/

**int** social;                   /\* 社保缴纳月数 \*/

**int** area;                    /\* 现有住房面积 \*/

**char** date[11];              /\* 申报日期 \*/

};

## 程序主框架设计

**int** main(**int** argc, **const** **char** \* argv[])

{

**int** m, T;

    cin >> m >> T;

    people \*person; // 指向所有报名人的基本资料首地址,通过调用函数getMess获取

**int** n;          // n为报名人数,通过调用函数getMess获取

    person = getMess(n);

    inssort<people, people>(person, n); //排序

**int** \*rank = **new** **int**[n];

//储存每个人的位次，方便后续找到同位次的申请者

    rank[0] = 1;

**for** (**int** i = 1; i < n;  i++)

    {

**……** //按排序后的顺序和题目要求得到所有人的位次

    }

    myOutput(person, m, T, n, rank); //查询并输出结果

**return** 0;

}

1. 函数getMess获取申请者的信息，存入结构体数组

people \*getMess(**int** &n) /\* 将文件数据读入内存 \*/

{

    FILE \*fp;

    fp = fopen("house.bin", "rb");

    fseek(fp, -1 \* (**long**)**sizeof**(**int**), 2);

    fread(&n, **sizeof**(**int**), 1, fp);

    rewind(fp);

    people \*tmp = **new** people[n];

    fread(tmp, **sizeof**(people), n, fp);

    fclose(fp);

**return** tmp;

}

1. inssort函数对结构体数组排序

**void** inssort (E \*A, **int** n)

{

**for** (从1～n)

    {

**bool** flag = 0;

**for** (从j = i ～ 0)

            {

**if** (住房面积 j < j - 1)

                {

                    swap (A, j, j - 1);

                    flag = 1;

                }

**else** **if** (住房面积相等)

                {

**if** (住房面积为0，刚性需求)

                    {

                        //先比社保缴纳月数再比申请日期

                    }

**else**

                    {

                        //改善性需求，直接比较申请日期

                    }

                }

**if** (!flag)

**break**;

            }

    }

}

1. myOutput函数，输出查询结果

**void** myOutput (people\* person, **int** m , **int** T, **int** n, **int** \*rank)

{

**int** tmpm = m; //最后名额的排名

**for** (**long** j = 0; j < T; j++)

    {

        string s;

        cin >> s; //      输入查询的身份证号

**for** (从0～n - 1)

        {

**if** (刚性需求且社保缴纳月数不超过2年)

                m++; //最后排名顺延一位

**if** (排名超出)

            {

                cout << "Sorry" << "\n";

                m = tmpm;

**break**;

            }

**if** (查询到对应的身份证号)

            {

**if** (刚性需求且社保缴纳月数不超过2年)

                    {

                        cout << "Sorry" << "\n";

                        m = tmpm;

**break**;

                    }

**long** A = 0, B = -1;

//A——后面同位次人数（包括自己，B——前面同位次的人数（不包括自己

**for** (**long** a = i; rank[a] == rank[i];a++)

                        A++;

**for** (**long** b = i; rank[b] == rank[i]; b--)

                        B++;

**long** C = A + B;         //同位次的总人数

**if** (C > 1)

                    {

**if** (人数超出剩余房源数量)

                        {

                            //输出x/C （C中选x人）

**break**;

                        }

**else**

                        {

                           //输出选房位次号区间

**break**;

                        }

                    }

**else** **if** (C == 1)

                    {

                        //直接输出位次

                    }

            }

        }

    }

}

## 样例设计与测试

样例测试一：正常数据

样例输入：

9 4

350102200609166049

250342323545313434

350102200609163286

110101196003074525

样例输出：

2

Sorry

3 4

2/3

## 六、分析与总结

1.掌握插入排序；

2.掌握文件操作；

# 题4：缺席考试的是谁？(ID: 16391)

## 一、问题描述

【问题描述】

程序设计考试结束了，传来个不好的消息：有一个学生没参加考试!需要尽快知道缺席考试的人是谁，以便尽快做出处理。

糟糕的是，尽管有签到表，但由于人数较多，签到情况比较混乱：有的签到表签在一张白纸上，有的虽然签在名册上，但并不是签在自己姓名旁，更有学生签到了别的签到表上……

现在只能根据这2n-1个姓名（名册上有n个学生姓名，签到有n-1个姓名，签到姓名和名册姓名可能混在一起了），来找到缺席考试的人是谁。唯一一个有利的条件是所有参加考试的人都签了名，且只签一次，签名也都正确无误。

现在任务交给你：编写一个程序，找出缺席考试的是谁。  
【输入形式】

有多组测试数据。

每组测试数据开始一行，是一个正整数n，表示总人数，n=0意味着输入结束并且不需要处理。

以下2n-1行，每行一个字符串，长度不超过20，表示一个人的姓名。姓名有大小写的英文字母、常用汉字组成(注意每个汉字占2个字节，中英文姓名都不排除有重名情况)。

40%的测试数据1 ≤ n≤ 10；

30%的测试数据1 ≤ n≤ 100；

20%的测试数据1 ≤ n≤ 103；

10%的测试数据1 ≤ n≤ 104；

提示：大量输入数据，C/C++输入推荐使用scanf函数

【输出形式】

对于每组测试数据，输出一行，只包含一个字符串，表示缺席的人的姓名。

【样例输入】

2

张三

张三

李四

0

【样例输出】

李四

## 二、问题分析及边界条件

要处理的问题：

1. 从输入的姓名中找出只出现过一次的人名（仅出现在花名册中）并输出；
2. 统计所有姓名的出现次数，输出出现次数为奇数个的姓名，即问题的解；

边界条件：

1.会出现有重名的情况，尽管一个姓名出现多次也不能确保这个姓名的同学没缺考；

2.姓名中有汉字，一个汉字占两个字节；

3.一次程序执行多组数据；

## 三、算法设计

通过上述分析，将采用下列方法解决相关问题：

1. 用pair型vector容器保存每个姓名和出现次数；
2. pair的第一、二个变量类型分别为string，int；
3. 遍历vecotr容器，寻找并输出出现奇数次的姓名；
4. 用while循环执行多组数据；

## 四、详细设计（从算法到程序）

1. 程序主结构设计

**int** main(**int** argc, **const** **char** \* argv[])

{

**int** N;

**while** (cin >> N && N) //多组数据，N为0时结束处理

    {

        vector<pair<string, **int**> > person;//保存姓名及出现次数

        string name; //保存输入的姓名

**for** (1 ~ 2\*N - 1) //有2n-1个姓名

        {

            ......//输入姓名

**if** (person为空)

               ......//第一个输入的姓名，存入vecotr中出现次数为1

**else**

            {

**while** (begin ~ end) //遍历vector

                {

**if** (当前姓名在vecotr中出现过)

                    {

                      ......//出现次数加1

**break**;

                    }

**if** (没出现过)

                    {

                        ......//存入vector中，出现次数为1

**break**;

                    }

                }

            }

        }

**while** (begin ~ end) //最后再遍历vector

        {

**if** (出现次数为奇数)

            {

                ......//输出姓名

**break**;

            }

        }

    }

**return** 0;

}

## 五、样例设计与测试

样例测试一:正常输入，没有重名，一组数据

样例输入：

5

张三

张三

李四

王五

李四

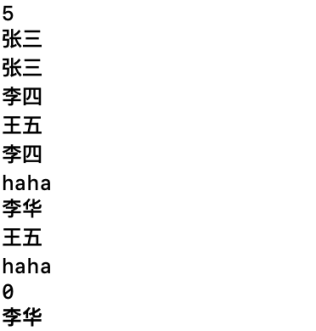
haha

李华

王五

haha

0



样例输出：**李华**

样例测试二:有重名，一组数据

样例输入：

5

张三

张三

李华

haha

李华

李华

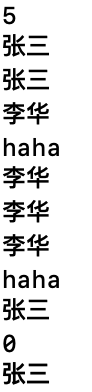
李华

haha

张三

0

样例输出：**张三**



样例输出三：有重名，多组数据

样例输入：

3

李华

张三

张三

张三

李华

5

MJ

李华

张三

李四

MJ

李四

李四

张三

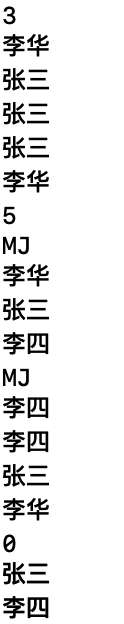
李华

0

样例输出：

**张三**

**李四**



## 六、分析与总结

1.掌握vector容器的使用；

2.掌握pair结构体模版的使用；

# 题5：数圈(ID: 16392)

## 一、问题描述

【问题描述】

以1为中心，用2,3,4, ..., n, ..., n\*n的数字围绕着中心输出数圈， 如若n=4，则

7 8 9 10

6 1 2 11

5 4 3 12

16 15 14 13  
【输入形式】

一个整数n(1<=n<=10)

【输出形式】

数圈矩阵

【样例输入】

5

【样例输出】

21 22 23 24 25

20 7 8 9 10

19 6 1 2 11

18 5 4 3 12

17 16 15 14 13

## 二、问题分析及边界条件

要处理的问题：

1. 找出数圈的输出规律；
2. 用一个二维数组模拟构建数圈，输出该二维数组；

边界条件：

1.以1为中心；

2.1 <= n <= 10;

## 三、算法设计

通过上述分析，将采用下列方法解决相关问题：

1. 使用二维数组模拟数圈；
2. 从1开始依次+1构建数圈；
3. 一共进行n次操作，第i步次操作处理步骤为：

1)当i是奇数

a)从当前数组位置向右前进i次，每次前进都在当前位置保存一个比上一个位置大1的数；

b)再从当前位置向下前进i次，每次前进都在当前位置保存一个比上一个位置大1的数；

1. i是偶数

a)从当前数组位置向左前进i次，每次前进都在当前位置保存一个比上一个位置大1的数；

b)再从当前位置向上前进i次，每次前进都在当前位置保存一个比上一个位置大1的数；

1. 输出二维数组；

## 四、详细设计（从算法到程序）

1. 开辟一个规模为n\*n的动态二维整型数组；
2. 用一个for循坏i = 1到 i = n，将n\*n个数依次保存在数组中正确位置；
3. 2中for循环中用if判断i的奇偶性，按照数圈规律，当i为奇数时，从上次保存数字在数圈（二维数组）中的位置开始，用两个for循环，且都循环i次，分别向右向下，保存比上一个位置大1的数，如果保存的数=n\*n时提前退出循环；当i为偶数时，也是两个for循环，方向为分别向左向上
4. 两个for循环输出二维数组；

## 五、样例设计与测试

样例测试一：正常输入

样例输入：2

样例输出：

**1 2**

**4 3**



样例测试二：取最大n = 10

样例输入：10

样例输出：

**73 74 75 76 77 78 79 80 81 82**

**72 43 44 45 46 47 48 49 50 83**

**71 42 21 22 23 24 25 26 51 84**

**70 41 20 7 8 9 10 27 52 85**

**69 40 19 6 1 2 11 28 53 86**

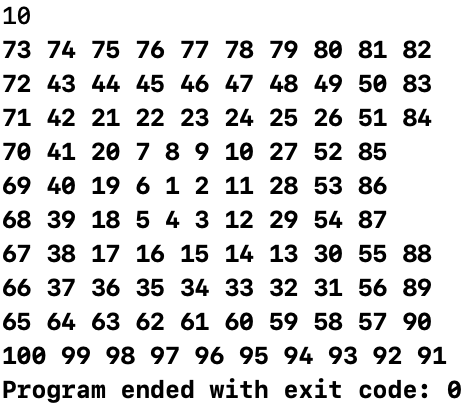
**68 39 18 5 4 3 12 29 54 87**

**67 38 17 16 15 14 13 30 55 88**

**66 37 36 35 34 33 32 31 56 89**

**65 64 63 62 61 60 59 58 57 90**

**100 99 98 97 96 95 94 93 92 91**



样例测试三：正常输入

样例输入：

7

样例输出：

**43 44 45 46 47 48 49**

**42 21 22 23 24 25 26**

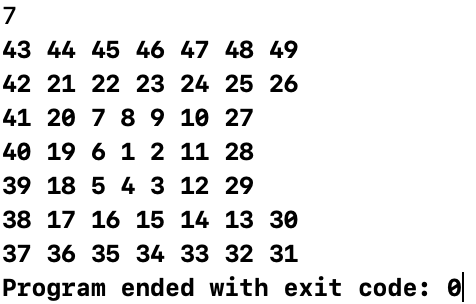
**41 20 7 8 9 10 27**

**40 19 6 1 2 11 28**

**39 18 5 4 3 12 29**

**38 17 16 15 14 13 30**

**37 36 35 34 33 32 31**



## 六、分析与总结

1.本问题需仔细观察数圈构造的规律，可将问题简化

# 题6：最大报销额(ID: 163414)

## 一、问题描述

【问题描述】现有一笔经费可以报销一定额度的发票。允许报销的发票类型包括买图书（A类）、文具（B类）、差旅（C类），要求每张发票的总额不得超过1000元，每张发票上，单项物品的价值不得超过600元。现请你编写程序，在给出的一堆发票中找出可以报销的、不超过给定额度的最大报销额。

【输入形式】测试输入包含若干测试用例。每个测试用例的第1行包含两个正数 Q 和 N，其中 Q 是给定的报销额度，N（N<=30）是发票张数。随后是 N 行输入，每行的格式为：  
      m Type\_1:price\_1 Type\_2:price\_2 … Type\_m:price\_m  
      其中正整数 m 是这张发票上所开物品的件数，Type\_i 和 price\_i 是第 i 项物品的种类和价值。物品种类用一个大写英文字母表示。当N为0时，全部输入结束，相应的结果不要输出。

【输出形式】对每个测试用例输出1行，即可以报销的最大数额，精确到小数点后2位。

【样例输入】

200.00 3  
2 A:23.50 B:100.00  
1 C:650.00  
3 A:59.99 A:120.00 X:10.00  
1200.00 2  
2 B:600.00 A:400.00  
1 C:200.50  
1200.50 3  
2 B:600.00 A:400.00  
1 C:200.50  
1 A:100.00  
100.00 0

【样例输出】

123.50  
1000.00  
1200.50

## 二、问题分析及边界条件

要处理的问题：

1. 如何保存一张支票的数据；
2. 找到能报销的支票以及该支票的额度；
3. 选择什么算法能达到问题最优解，即最大报销额；

边界条件：

1. 输入中包含‘ ：’；
2. 有多组输入；
3. 每张支票的总额不超过1000元；
4. 每张支票中当件物品不超过600元；
5. 精确到小数点后两位；

## 三、算法设计

通过上述分析，将采用下列方法解决相关问题：

1. 用long double型的vector容器保存每张支票的总金额；
2. 用bool型数组判断每张支票是否可报销；
3. 用一个char型变量保存输入中‘ ：’；
4. 用while循环实现多组数据的输入；
5. 用动态规划和递归的算法求最大报销额；
6. 引用头文件iomanip.h中fix和setprecision函数输出时精确到小数点后两位；

## 四、详细设计（从算法到程序）

1.递归函数Maxexp用于实现动态规划算法：

**long** **double** Maxexp (**int** N, **long** **double** left, **bool**\* flag, vector<**long** **double**> checksum, **int** i)

//N为支票数，left为当前剩余可报销额，数组flag判断对应支票是否可报//销，checksum保存每张支票的总金额，i为当前要报销的支票，从0开始

{

**if** (i == 支票数 - 1)

    {

**if** (剩余可报销额 < 当前支票总金额 || 当前支票不可报销)

**return** 0;

**else**

        {

**return** checksum[i];//当前支票总金额

        }

    }

**if** (剩余可报销额 < 当前支票总金额 || 当前支票不可报销)

    {

...... //当前支票不可报销，进入下一次递归判断下一张支票

}

**else**

{

......

//继续递归，取不报销该支票的情况和报销该支票的情况中的最大值

}

   }

2.程序主函数设计：

**int** main(**int** argc, **const** **char** \* argv[])

{

**long** **double** Q, SUM = 0;

**int** N;

**while** (cin >> Q >> N && N)

//输入支票数，可报销的总金额，支票数为0时结束

    {

        vector<**long** **double**> checksum;

//定义保存支票金额的vecotr容器

**bool** \*flag = **new** **bool**[N];

//定义判断支票是否可报销的bool型数组

        checksum.resize(N,0); //初始化

**for** (0 ～ N - 1)

        {

            ......//输入支票包含物品数

**for** (0 ~ num)

            {

                ...... //输入物品类型，：，物品金额

**if** (支票不可报销)

                 {

...... //bool数组中对应元素置为1

}

                checksum[i] += amount;

//统计当前支票总金额

            }

        }

        SUM = Maxexp(N, Q, flag, checksum, 0); //递归

        cout << fixed << setprecision(2) << SUM << "\n";

//输出

    }

**return** 0;

}

## 五、样例设计与测试

样例测试一：仅有一组数据，不存在不可报销的支票

样例输入：

1000.00 3

2 A:300.00 C:29.99

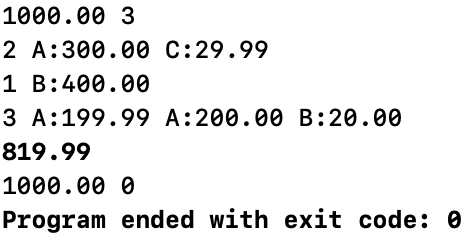
1 B:400.00

3 A:199.99 A:200.00 B:20.00

1000.00 0

样例输出：

**819.99**



样例测试二：多组组数据，存在不可报销的支票

样例输入：

1000.00 3

2 A:300.00 C:29.99

1 B:400.00

3 A:199.99 A:200.00 B:20.00

1200.00 5

3 A:100.00 F:300.00 B:20.00

2 A:700.00 B:200.00

3 A:100.00 B:400.00 C:500.00

2 C:300.00 A:500.00

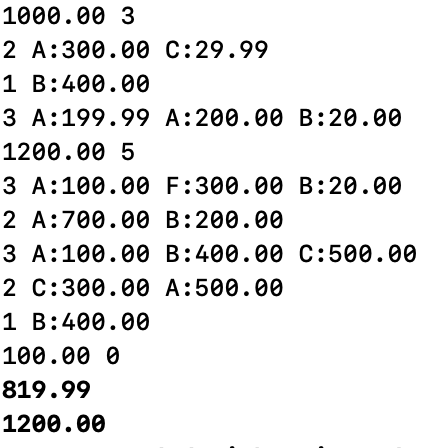
1 B:400.00

100.00 0

样例输出

**819.99**

**1200.00**



样例测试三：正常输入

样例输入：

200.00 2

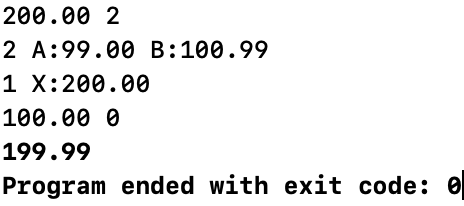
2 A:99.00 B:100.99

1 X:200.00

100.00 0

样例输出:

**199.99**



## 分析与总结

1. 掌握动态规划和递归；
2. 掌握vecotr容器的使用

# 题7：公交系统(ID: 16402)

## 一、问题描述

【问题描述】

        城市公交系统有一个记录仪，用于记录每个站点的乘客人数的变化情况，例如：x表示到站前公交车上的乘客人数，y表示离站时公交车上的乘客人数，则该记录仪记录的该站的数字为y-x。

        对于一辆公交车和n个车站，a1,a2,...,an为该公交车在各站的记录数据。

        假定w为该公交车可容纳的最大乘客人数，编程求出在第一站停靠之前公交车上人数的可能数据有多少种？

【输入形式】

        第一行包含两个数据n和w(1<=n<=1000, 1<=w<=109)，分别表示车站的数目和公交车可容纳的最大乘客人数。

        第二行包含一个序列a1,a2,...,an，表示记录仪记录的各站的数据。

【输出形式】

        输出一个整数，表示公交车在第一站停靠之前可能的乘客人数数据的个数，如果没有，则输出0。

【样例输入1】

3 5

2 1 -3

【样例输出1】

3

【样例输入2】

2 4

-1 1

【样例输出2】

4

【样例输入3】

4 10

2 4 1 2

【样例输出3】

2

【样例说明】

在第一个样例中，乘客数可能有0、1、2，共3种情况

在第二个样例中，乘客数可能有1、2、3、4，共4种情况

在第三个样例种，乘客数可能为0或1，共2种情况

## 二、问题分析及边界条件

要处理的问题：

1.保存个车站记录数据；

2.从所给的各车站的记录数据中找出可能增加的最多人数和减少的最多人数

3.确定开始时车上的最少人数和最多人数；

边界条件：

1.可能有0种情况；

2.车上人数不能小于0或大于最大可容纳人数；

## 三、算法设计

通过上述分析，将采用下列方法解决相关问题：

1. 用int型vector容器保存经过车站后变化的人数；
2. 用两个整型变量记录增加的最大人数和减少的最大人数；
3. 遍历vecotr容器，确定增加最大人数和减少的最大人数；
4. 用一个整型变量保存经过所有车站后变化的总人数；
5. 开始时车上最少人数为减少的最大人数的相反数，车上最大人数为最大可容纳人数 - 增加的最大人数；
6. 输出可能情况，即车上最大人数 - 车上最小人数 + 1；

## 四、详细设计（从算法到程序）

1.程序主结构设计

**int** main(**int** argc, **const** **char** \* argv[])

{

    ...... //输入车站数目n， 最大容纳人数w

vector<**int**> a;

//保存经过连续车站后的人数变化，a[0]保为经过第一个车站变化的人数

//a[1]为经过第一、二号车站后变化的人数

//a[2]为经过第一、二、三号车站后变化的人数......

**long** Max = -1, Min = 999999;

//定义并初始化增加的最大人数和减少的最大人数

**int** change\_sum = 0;

//定义经过所有车站后变化的人数

**for** (1 ～ n)

    {

        //输入1～n个车站的记录数据

.......//记录a[i - 1],并更新Max和Min

//更新change\_sum

    }

**if** (change\_sum的绝对值大与最大可容纳人数w)

{.......} //输出0，结束

**long** up = w, down = 0;

//定义上界（初始最大人数）和下界（初始最小人数）,并初始化

**if** (增加的最大人数为正数)

{

...... //更新上界

}

**if** (减少的最大人数为负数)

{

...... //更新下界

}

**long** sum = 0;

//定义可能的初始情况种数

**if** (上界 >= 下界)

        sum = up - down + 1;

    cout << sum << "\n";

**return** 0;

}

## 五、样例设计与测试

## 样例测试一：Min > 0

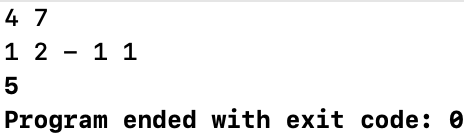
样例输入：

4 7

1 2 - 1 1

样例输出：

**5**



样例测试二：Max < 0

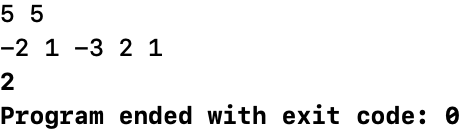
样例输入：

5 5

-2 1 -3 2 1

样例输出：

**2**



样例测试三：正常输入

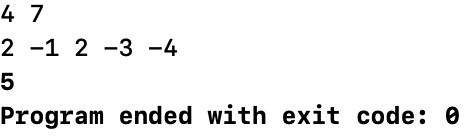
样例输入：

4 7

2 -1 2 -3 -4

样例输出：

**5**



## 六、分析与总结

1.仔细分析题目，找到上界和下界可快速求得可能情况数；

# 题8：Web导航(ID: 16419)

## 一、问题描述

【问题描述】

标准的Web浏览器具有在最近访问的页面中前后移动的特性。实现这些特性的一种方法是使用两个堆栈来跟踪可以通过前后移动到达的页面。在这个问题中，我们要求实现这一点。

       需要支持以下命令：

       BACK：将当前页面压入前向堆栈的顶部；从后向堆栈的顶部弹出该页，使其成为新的当前页。如果后向堆栈为空，则该指令忽略。

       FORWARD：将当前页面压入后向堆栈的顶部；从前向堆栈的顶部弹出该页，使其成为新的当前页。如果前向堆栈为空，则该指令忽略。

       VISIT：将当前页面压入后向堆栈的顶部，将URL指定为新的当前页。前向堆栈被清空。

       QUIT：退出浏览器。

       假设浏览器最初在网址http://www.game.org/上加载网页。

【输入形式】输入是一个命令序列。命令关键字BACK、FORWARD、VISIT和QUIT都是大写。URL中无空格，最多有70个字符。假定在任何时候，每个堆栈中没有问题实例需要超过100个元素。输入的结尾由QUIT命令标识。

【输出形式】除QUIT外的每个命令，如果命令没有被忽略，则在命令执行后输出当前页面的URL，否则，打印"Ignored"。每个命令的输出独立打印一行。QUIT命令无输出。

【样例输入】

VISIT http://game.ashland.edu/  
VISIT http://game.baylor.edu/acmicpc/  
BACK  
BACK  
BACK  
FORWARD  
VISIT http://www.our.com/  
BACK  
BACK  
FORWARD  
FORWARD  
FORWARD  
QUIT

【样例输出】

http://game.ashland.edu/

http://game.baylor.edu/acmicpc/

http://game.ashland.edu/

http://www.game.org/

Ignored

http://game.ashland.edu/

http://www.our.com/

http://game.ashland.edu/

http://www.game.org/

http://game.ashland.edu/

http://www.our.com/

Ignored

## 二、问题分析及边界条件

要处理的问题：

1.如何保存前向堆栈和后向堆栈的页面和当前页面；

边界条件：

1. 命令关键字都是大写；
2. 浏览器最初在网址http://www.game.org/上加载网页；
3. 有多组输入，输入为“QUIT”时结束；

## 三、算法设计

通过上述分析，将采用下列方法解决相关问题：

1. 用两个string型的栈作为前向堆栈和后向堆栈；
2. 用一个string型变量保存当前页面，初始化为http://www.game.org/；
3. 用while循环实现多行命令关键字输入；
4. 根据输入的关键字执行不同操作，每次正确操作都输出当前页面，错误操作这输出“Ignored”；

## 详细设计（从算法到程序）

1.程序主结构设计

**int** main(**int** argc, **const** **char** \* argv[])

{

string command;

//定义命令关键字

    stack<string> front;

stack<string> back;

//定义前向堆栈和后向堆栈

string cur = "http://www.game.org/";

//当前页面

**while** (输入关键字 当关键字为“QUIT”时退出)

    {

**if** (关键字为“VISIT”)

        {

           ......

//输入URL，将当前页面压入后向堆栈，将URL指定为新的当前页。清空前向堆栈

        }

**else** **if** (关键字为“BACK”)

        {

**if** (后向堆栈为空)

            {

                ...... //输出Ignored

**continue**;

            }

            ......

//后向堆栈弹出堆顶页面，更新当前页面和前向堆栈

        }

**else**

        {

**if** (前向堆栈为空)

            {

                ...... //输出Ignored

**continue**;

            }

            ......

//前向堆栈弹出堆顶页面，更新当前页面和后向堆栈

        }

        ...... //输出当前页面

    }

**return** 0;

}

## 样例设计与测试

样例测试一：正常输入

样例输入：

VISIT http://game.baylor.edu/acmicpc/

VISIT http://www.our.com/

BACK

BACK

BACK

FORWAR**D**

FORWARD

VISIT http://game.ashland.edu/

FORWARD

QUIT

样例输出：

**http://game.baylor.edu/acmicpc/**

**http://www.our.com/**

**http://game.baylor.edu/acmicpc/**

**http://www.game.org/**

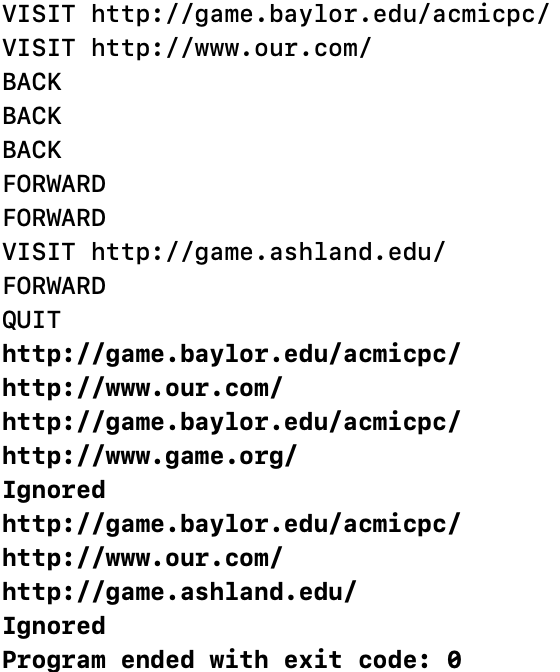
**Ignored**

**http://game.baylor.edu/acmicpc/**

**http://www.our.com/**

**http://game.ashland.edu/**

**Ignored**



样例测试二：正常输入

样例输入：

VISIT http://game.baylor.edu/acmicpc/

VISIT http://www.our.com/

BACK

BACK

BACK

FORWAR**D**

FORWARD

VISIT http://game.ashland.edu/

FORWARD

BACK

BACK

FORWARD

VISIT http://hnu/

VISIT <http://202.197.98.89/>

QUIT

样例输出：

**http://game.baylor.edu/acmicpc/**

**http://www.our.com/**

**http://game.baylor.edu/acmicpc/**

**http://www.game.org/**

**Ignored**

**http://game.baylor.edu/acmicpc/**

**http://www.our.com/**

**http://game.ashland.edu/**

**Ignored**

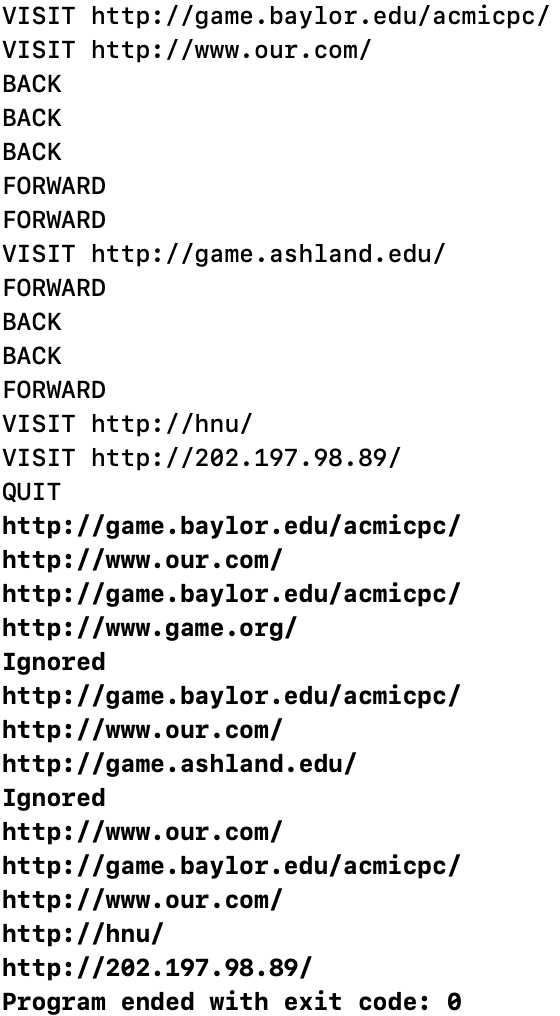
**http://www.our.com/**

**http://game.baylor.edu/acmicpc/**

**http://www.our.com/**

**http://hnu/**

**http://202.197.98.89/**



样例测试三：正常输入

样例输入：

VISIT http://game.baylor.edu/acmicpc/

VISIT http://www.our.com/

BACK

BACK

BACK

VISIT http://www.steam.com/

FORWAR**D**

FORWARD

VISIT http://game.ashland.edu/

FORWARD

BACK

BACK

QUIT

样例输出：

**http://game.baylor.edu/acmicpc/**

**http://www.our.com/**

**http://game.baylor.edu/acmicpc/**

**http://www.game.org/**

**Ignored**

**http://www.steam.com/**

**Ignored**

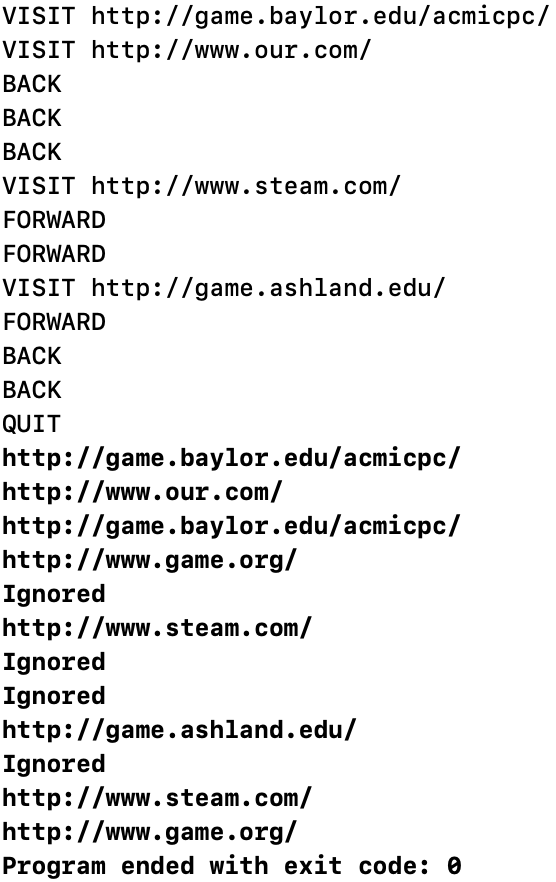
**Ignored**

**http://game.ashland.edu/**

**Ignored**

**http://www.steam.com/**

**http://www.game.org/**



## 六、分析与总结

1.掌握stack的使用；

# 题9：[相等的多项式](http://202.197.98.89/assignment/programList.jsp?proNum=8&assignID=949)(ID:16438)

## 一、问题描述

****问题描述****

小明现在在学习多项式的展开：就是把一个形如

（x+a1）（x+a2） ... （x+an）

展开成如下形式：

xn + b1xn-1 + b2xn-2 + ... + bn-1x + bn

比如 （x+1）（x+2）= x2 + 3x + 2

      (x+1)3 = x3 +3x2 +3x + 1

小明做了很多练习，但是不知道对错，现在请求你的帮助，判断小明的展开式是否正确。

****输入格式****

有多组测试数据。

每组测试数据有三行，第一行是一个正整数N，表示多项式最高指数。N=0表示输入结束，并且不需要处理。

第二行N个整数ai，用空格隔开，i=1，...,N(-100≤ai≤100)

第三行N个整数bi，用空格隔开，i=1，...,N，(-109≤bi≤109)

40%的测试数据 1 ≤ N < 5；

30%的测试数据 5 ≤ N < 10；

20%的测试数据10 ≤ N < 15；

10%的测试数据 15 ≤N≤ 20；

****输出格式****

　对于每组测试数据，输出一行一个字符‘Y'如果展开式是正确的，输出‘N’如果展开式错误。

****样例输入****

2

1 2

3 2

3

1 1 1

3 3 1

4

0 0 0 1

0 0 0 1

0

****样例输出****

Y

Y

N

## 二、问题分析及边界条件

要处理的问题：

1. 展开多项式时如何计算每项的常数值；
2. 保存计算得到的常数值与小明计算的常数值对比，判断小明计算的是否正确

边界条件：

1.有多组数据；

## 三、算法设计

通过上述分析，将采用下列方法解决相关问题：

1. 用回溯法，按照数学计算的方法，在每个括号中取一项计算，穷举所有情况，如：（x+a1）（x+a2）（x+a3）（x+a4）选取方式有：0000，1000，0100，0010，0001……（1表示选an，0表示选x）

## 详细设计（从算法到程序）

1. 全局变量设置

**int** N; //多项式最高指数

vector<**int**> a; //an

vector<**int**> b; //bn

**long** c[30]; //cn

1. 函数主结构设计

**int** main(**int** argc, **const** **char** \* argv[])

{

**while** (输入N,且N不为0)

    {

        ...... //每次输入新的数据都要初始化an,bn,c

**for** (1~N)

        {

            ......//输入an

        }

**for** (1~N)

        {

            ......//输入bn

        }

        spread(0, 0, 1); //调用递归函数spread

**bool** flag = 1; //判断多项式是否相等

**for** (0 ～ N - 1)

        {

**if** (b[n] 与 c[n+1] 不相等)

            {

                ...... //flag置为0，退出循环

            }

        }

**if** (flag为1)

            cout << "Y\n";

**else**

            cout << "N\n";

    }

**return** 0;

}

1. 函数spread用于模拟展开多项式的过程

**void** spread (**int** n, **int** num\_a, **long** sum)

//已选择的x和a的个数，已选择的a的个数，计算的常数值

{

**if** (n == N)

    {

        ...... //保存最终计算得到的常数值到c中相应位置上，返回

    }

    spread(n + 1, num\_a, sum); //不选此括号内的a

    spread(n + 1, num\_a + 1, sum \* a[n]); //选此括号内的a

}

## 五、样例设计与测试

## 样例测试一：正常输入

样例输入：

4

2 1 3 1

7 17 17 6

3

1 1 1

3 3 1

4

1 1 1 1

4 3 2 1

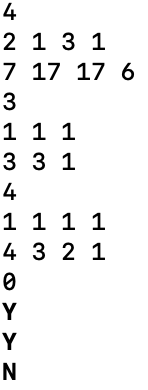
0

样例输出：

**Y**

**Y**

**N**



样例测试二：an值较大

样例输入：

3

99 98 100

297 29402 970200

4

97 98 99 100

394 58211 3822194 94109400

4

97 98 99 100

1 1 1 1

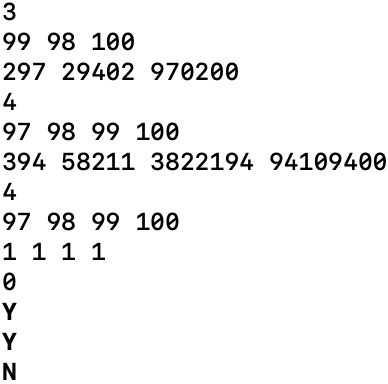
0

样例输出：

**Y**

**Y**

**N**



样例测试三：an存在负数

样例输入：

3

-1 1 2

2 -1 -2

3

-1 -1 2

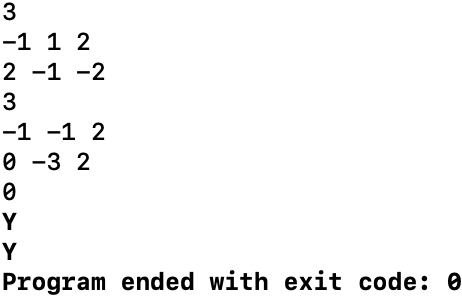
0 -3 2

0

样例输出：

**Y**

**Y**



## 六、分析与总结

1.掌握递归函数的使用；

# 题10：疫情期间(ID: 16430)

## 一、问题描述

【问题描述】

        正值新冠疫情期间，阿迪没法返回学校学习，他希望通过参加一些比赛来提高一下编程技能，同时做做运动。他收集了接下来的 **n**天里每一天的信息，包括健身房是否开放，或者互联网上是否有程序设计竞赛。

        第**i**天可以有以下四种情况之一：

该天健身房不开放，互联网上也没有竞赛

该天健身房不开放，但互联网上有竞赛

该天健身房开放，但互联网上没有竞赛

该天健身房开放，互联网上也有竞赛

        每天阿迪要么休息，要么编写程序（如果该天有竞赛），要么做运动（如果该天健身房开放）。

        现在有一个限制条件：不能连续两天都去做运动，或者连续两天都编写程序。阿迪对自己要求很高，希望尽量多写程序或者多做运动，使得休息的天数尽量最少，求出这个天数。

【输入形式】

        输入的第一行为一个正整数 ****n**** (1≤ ****n**** ≤ 100)，表示接下来的天数。

        第二行为一个用空格分隔的整数序列 ****a****1、****a****2、…、****a****n（0≤****a****i≤3），这里

****a****i=0，第 ****i**** 天健身房不开放，互联网上也没有竞赛

****a****i=1，第 ****i**** 天健身房不开放，但互联网上有竞赛

****a****i=2，第 ****i**** 天健身房开放，但互联网上没有竞赛

****a****i=3，第 ****i**** 天健身房开放，互联网上也有竞赛

【输出形式】

 输入阿迪可能休息的最小天数。注意限制条件：

* 不能连续两天去做运动
* 不能连续两天编写程序

【样例输入1】

4

1 3 2 0

【样例输出1】

2

【样例输入2】

7

1 3 3 2 1 2 3

【样例输出2】

0

【样例输入3】

2

2 2

【样例输出3】

1

【样例说明】

在第一个样例中，阿迪在第一天编写程序，在第三天做运动，因此他仅有两天可以休息。

在第二个样例中，阿迪可以在第1、3、5、7天编写程序，其他天做运动，因此没有哪天休息。

在第三个样例中，阿迪可以在第1天或第2天做运动，但不能连续两天运动，因此他有一天休息。

## 二、问题分析及边界条件

要处理的问题：

1. 如何选择能让休息天数最少；
2. 采用何种方式保存每天健身房和竞赛的情况，便于后续查询；
3. 采用何种方式保存阿迪某一天的活动情况；

边界条件：

1.不能连续两天做同样的活动；

## 三、算法设计

通过上述分析，将采用下列方法解决相关问题：

1. 用贪心法，在碰到一天健身房和竞赛都可参加的一天，选择可让最终休息天数最少的活动，其他情况能不休息就不休息；
2. 用整型数组保存每天活动情况；
3. 用一个整型变量保存，昨天阿迪的活动情况，0为休息，1为竞赛，2为健身房；

## 四、详细设计（从算法到程序）

## 1.程序主结构设计

**int** main(**int** argc, **const** **char** \* argv[])

{

     ......         //输入活动总天数

**int** \*sit = **new** **int**[D + 1]; //sit保存每天健身房和竞赛的情况

**for** (1 ~ D)

      ...... //输入每天活动情况

**int** rest = 0; //rest保存休息天数

    rest = minrest(D, 1, sit, 0); //调用minrest函数

    cout << rest << "\n"; //输出rest

**return** 0;

}

1. 递归函数minrest

**int** minrest (**int** D, **int** d, **int** \*sit, **int** yes)

//总天数,目前天数,sit数组,昨天活动情况

{

**if** (d == D)

    {

**if** (今天没有活动)

**return** 1; //只能休息，返回1

**if** (昨天休息 或 昨天有两种活动)

**return** 0; //不休息，返回0

**else**

        {

**if** (今天活动和昨天所做的活动不同)

**return** 0;

**else**

**return** 1;

        }

    }

**if** (今天没有活动)

        ...... // 只能休息

**else** **if** (今天有两个活动)

    {

**int** tmp1;

**if** (昨天休息)

...... //今天选健身或比赛，最终取休息天数最少的

**else** **if** (昨天比赛)

            ...... //则今天去健身

**else**

            ...... //今天去比赛

**return** tmp1;

    }

**else**

    {

**if** (昨天休息)

            ...... //则今天不休息

**else** **if** (今天活动和昨天不同)

            ...... //则今天不休息

**else**

            ......     //今天活动和昨天相同，只能休息

    }

}

## 五、样例设计与测试

## 样例测试一：正常输入

## 样例输入：

## 5

3 3 3 3 3

样例输出：

**0**



## 样例测试二：正常输入

## 样例输入：

7

1 3 2 0 1 2 3

样例输出:

**2**

：

## 样例测试三：正常输入

## 样例输入：

50

3 1 0 0 1 2 3 4 1 2 3 0 3 2 1 1 2 2 3 3 2 1 1 0 0 1 2 2 2 3 3 3 0 0 1 2 3 2 1 3 2 1 0 1 2 2 1 2 3 1

样例输出：

**15**



## 六、分析与总结

1.掌握递归函数的使用；

2.掌握贪心算法；