

1 字符串最后一个单词的长度

```
#include <iostream>
#include <string>
using namespace std;

int main(){
    string str;
    getline(cin,str);
    int count=0;
    for (int i=str.size()-1;i>=0;i--){
        if(str[i]!=' '){
            count++;
        }
        else{break;}
    }
    cout<< count << endl;
}
```

2 计算某字母出现的次数

```
#include<iostream>
#include<string>
using namespace std;
int main()
{
    string str;
    char s;
    int count=0;
    //    cin>>str; //遇“空格”“tab”“回车”结束
    getline(cin, str, '\n');// 默认'\0'
    cin >> s;
    for(int i=0;i<str.size();i++)
    {
        if(str[i] == s||str[i]+32 == s||str[i]-32 == s)
        {
            count++;
        }
    }
    cout<<count<<endl;
    return 0;
}
```

3 明明的随机数

明明想在学校中请一些同学一起做一项问卷调查，为了实验的客观性，他先用计算机生成了N个1到1000之间的随机整数（N≤1000），对于其中重复的数字，只保留一个，把其余相同的数去掉，不同的数对应着不同的学生的学号。然后再把这些数从小到大排序，按照排好的顺序去找同学做调查。请你协助明明完成“去重”与“排序”的工作(同一个测试用例里可能会有多组数据(用于不同的调查)，希望大家能正确处理)。

```

#include<iostream>
#include<set>
using namespace std;
int main(){
    int N,n;
    set<int> ss;
    while(cin>>N){
        ss.clear();
        while(N--){
            cin>>n;
            ss.insert(n);
        }
        for(set<int>::iterator iter=ss.begin(); iter != ss.end();iter++){
            cout << (*iter)<< endl;
        }
    }
    return 0;
}

```

4 字符串分割

- 连续输入字符串，请按长度为8拆分每个字符串后输出到新的字符串数组；
- 长度不是8整数倍的字符串请在后面补数字0，空字符串不处理。

```

#include<iostream>
using namespace std;
int main(){
    string str;
    while(getline(cin,str)){
        while(str.size()>8){
            cout<< str.substr(0,8) <<endl;
            str = str.substr(8);
        }
        cout<< str.append(8-str.size(),'0')<<endl;
    }
}

```

5 十六进制的数，该数值的十进制表示

```

#include<iostream>
#include<string>
#include<cmath>
using namespace std;
int main(){
    string s;
    while(cin>>s){
        int bit = 0;
        int ans = 0;
        for (int i=s.length()-1;i>1;i--){
            if(s[i]>='0'&&s[i]<='9')
                ans+=(s[i]-'0')*pow(16,bit++);
            else if(s[i]>='A'&& s[i]<='F')
                ans += (s[i]-'A'+10)*pow(16,bit++);
        }
    }
}

```

```

        cout<< ans <<endl;
    }
    return 0;
}

```

6 质数因子

功能:输入一个正整数, 按照从小到大的顺序输出它的所有质因子 (重复的也要列举) (如180的质因子为2 2 3 3 5) 最后一个数后面也要有空格

```

#include <iostream>
#include <math.h>
using namespace std;
int main(){
    long long input;
    while(cin>>input){
        for (int i=2;i<=sqrt(input);i++){////sqrt 防止时间过长
            while (input % i == 0){
                cout<< i<<' ';
                input = input/i;
            }
        }
        if (input>1) cout<< input <<' ';
    }
    //    return 0;
}

```

7 取近似值

写出一个程序, 接受一个正浮点数值, 输出该数值的近似整数。如果小数点后数值大于等于5,向上取整; 小于5, 则向下取整。

```

#include<iostream>
using namespace std;
int main(){
    double num;
    cin>>num;
    cout<< int(num+0.5)<<endl; ////加0.5取整
    return 0;
}

```

8 合并表记录

数据表记录包含表索引和数值 (int范围的正整数), 请对表索引相同的记录进行合并, 即将相同索引的数值进行求和运算, 输出按照key值升序进行输出。

```

#include <iostream>
#include <map>
using namespace std;
int main(){
    map<int,int> my_map;
    int key, value , num;
    cin>> num;
}

```

```

while(num-- && cin>> key >> value){
    my_map[key] += value;////将相同的记录合并
}
for (auto iter=my_map.begin();iter!=my_map.end();iter++){
    cout<< iter->first << " " << iter->second << endl;
}
return 0;
}

```

9 提取不重复的整数

输入一个int型整数，按照从右向左的阅读顺序，返回一个不含重复数字的新的整数。保证输入的整数最后一位不是0。例：输入：9876673，输出：37689。

```

#include<iostream>
#include<vector>
using namespace std;
int main(){
    int num;
    int n;
    vector<int> hash(10,0);
    cin>> num;
    while(num>0){
        n = num%10;
        num = num/10;
        if(hash[n]==0){
            hash[n]=1;
            cout<< n;
        }
    }
    return 0;
}
////hash表是怎么实现的

```

10 字符个数统计

编写一个函数，计算字符串中含有的不同字符的个数。字符在ASCII码范围内(0~127)，换行表示结束符，不算在字符里。不在范围内的不作统计。多个相同的字符只计算一次。例如，对于字符串abaca而言，有a、b、c三种不同的字符，因此输出3。

```

#include<bits/stdc++.h> ///头文件
using namespace std;
int main(){
    string str;
    cin >> str;
    unordered_set<char> set;
    for (char c: str)
        if (c>=0 && c<=127) set.insert(c);
    cout << set.size() << endl;
    return 0;
}

```

11 数字颠倒

输入一个整数，将这个整数以字符串的形式逆序输出（程序不考虑负数的情况，若数字含有0，则逆序形式也含有0，如输入为100，则输出为001）

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    int num;
    cin>> num;
    string str = to_string(num);
    reverse(str.begin(),str.end());
    cout << str << endl;
    return 0;
}

int main(){
    int n;
    while(cin >> n){
        string ans;
        do{
            ans += n%10+'0'; //不用to_string的情形
            n /= 10;
        }while(n);
        cout << ans;
    }
    return 0;
}
```

12 字符串反转

```
#include<bits/stdc++.h>
using namespace std;
int main () {
    string str;
    cin >> str;
    // reverse(str.begin(),str.end());
    // cout << str;
    for(int i=str.length()-1;i>=0;i--){
        cout<< str[i]; //输出的方式不同
    }
    return 0;
}
```

```
#include <stdio.h>
#include <string.h>
int main(){
    char a[1000]={};
    int i=0, j=0;
    char temp;
    scanf("%s",a);
    int len = strlen(a);
    j = len -1 ;
    while (i<j){
        temp = a[i];
        a[i] = a[j];
        a[j] = temp;
    }
```

```

        i++;
        j--;
    }
    for(i=0; i<len; i++){
        printf("%c",a[i]);
    }
    printf("\n");
    return 0;
}

```

13 句子逆序

将一个英文语句以单词为单位逆序排放。例如“I am a boy”，逆序排放后为“boy a am I”。所有单词之间用一个空格隔开，语句中除了英文字母外，不再包含其他字符。

```

#include<bits/stdc++.h>
#include<iostream>
#include<string>
using namespace std;
int main(){
    string s;
    while(getline(cin,s)){
        int n=s.size();
        string tmp;
        for(int i=n-1;i>=0;i--){
            if(s[i]!=' '){
                tmp = s[i] + tmp;
            }
            else{
                cout<< tmp << ' ';
                tmp.clear();
            }
        }
        cout << tmp <<endl;
    }
}

// int main() {
//     string str, res;
//     while(cin >> str) {
//         str += " " +res;
//         res = str;
//     }
//     cout << res << endl;
//     return 0;
// }/////每当有空格就停止了

```

106 字符逆序

tneduts a ma I

```
#include<bits/stdc++.h>
using namespace std;
int main() {
    string str;
    getline(cin, str);
    reverse(str.begin(),str.end());
    cout<<str<<endl;
    return 0;
}
```

14 字符串排序

给定n个字符串，请对n个字符串按照字典序排列。

```
#include<bits/stdc++.h>
using namespace std;

int main() {
    int n; cin >> n;
    vector<string> res(n);
    for(int i = 0; i < n; i++) cin >> res[i];
    sort(res.begin(), res.end());
    for(string s : res) cout << s << endl;
    return 0;
} //simple writing
```

15 int型数据在内存中存储时1的个数

```
#include<iostream>
using namespace std;
int main()
{
    int n;
    cin >> n;
    int count=0;
    while(n){
        count++;
        n = (n-1)&n;
    }
    cout << count<< endl;
    return 0;
}
```

16 购物单（动态规划）

题目描述

王强今天很开心，公司发给N元的年终奖。王强决定把年终奖用于购物，他把想买的物品分为两类：主件与附件，附件是从属于某个主件的，下表就是一些主件与附件的例子：

主件	附件
电脑	打印机，扫描仪
书柜	图书
书桌	台灯，文具
工作椅	无

如果要买归类为附件的物品，必须先买该附件所属的主件。每个主件可以有 0 个、1 个或 2 个附件。附件不再有从属于自己的附件。王强想买的东西很多，为了不超出预算，他把每件物品规定了一个重要度，分为 5 等：用整数 1~5 表示，第 5 等最重要。他还从因特网上查到了每件物品的价格（都是 10 元的整数倍）。他希望在不超过 N 元（可以等于 N 元）的前提下，使每件物品的价格与重要度的乘积的总和最大。

设第 j 件物品的价格为 $v[j]$ ，重要度为 $w[j]$ ，共选中了 k 件物品，编号依次为 j_1, j_2, \dots, j_k ，则所求的总和为：

$v[j_1]*w[j_1]+v[j_2]*w[j_2]+ \dots +v[j_k]*w[j_k]$ 。（其中 * 为乘号）

请你帮助王强设计一个满足要求的购物单。

输入：输入的第 1 行为两个正整数，用一个空格隔开：N m（其中 N（ <32000 ）表示总钱数，m（ <60 ）为希望购买物品的个数。）从第 2 行到第 m+1 行，第 j 行给出了编号为 j-1 的物品的基本数据，每行有 3 个非负整数 v p q（其中 v 表示该物品的价格（ $v<10000$ ），p 表示该物品的重要度（1~5），q 表示该物品是主件还是附件。如果 q=0，表示该物品为主件，如果 q>0，表示该物品为附件，q 是所属主件的编号）

输出：输出文件只有一个正整数，为不超过总钱数的物品的价格与重要度乘积的总和的最大值（ <200000 ）。

```
#include <iostream>
#include <string>
using namespace std;

int getMax(int x , int y){return x>y?x:y;}

int main(){
    int N,m;
    int weight[60][3]={0}; //价格
    int value[60][3]={0};
    int dp[60][3200]={0}; // /=0

    cin >> N >> m;
    N /=10;

    //清单
    for(int i=1;i<=m;i++){
        int v,p,q;
        cin >> v >> p >> q; //单件价格，价值，主件or附件
        v /=10;
        if(q==0){weight[i][0]=v;value[i][0]= v*p;}
        else {
            if(weight[q][1]==0){weight[q][1]=v;value[q][1]=v*p;}
            else{weight[q][2]=v;value[q][2]=v*p;}
        }
    }
}
```



```

//traverse
for(int i=1; i<=m; i++){
    for(int j=1; j<=N; j++){
        dp[i][j]=dp[i-1][j];
        if(j>=weight[i][0])
            dp[i][j]= getMax(dp[i][j], dp[i-1][j-weight[i][0]]+value[i][0]);
        if(j>=weight[i][0]+weight[i][1])
            dp[i][j]= getMax(dp[i][j], dp[i-1][j-weight[i][0]-weight[i][1]]+value[i][0]+value[i][1]);
        if(j>=weight[i][0]+weight[i][2])
            dp[i][j]= getMax(dp[i][j], dp[i-1][j-weight[i][0]-weight[i][1]-weight[i][2]]+value[i][0]+value[i][1]+value[i][2]);
        if(j>=weight[i][0]+weight[i][1]+weight[i][2])
            dp[i][j]= getMax(dp[i][j], dp[i-1][j-weight[i][0]-weight[i][1]-weight[i][2]]+value[i][0]+value[i][1]+value[i][2]);

        //          int t;
        //          dp[i][j] = dp[i-1][j];
        //          if(j>=weight[i][0]){t=dp[i-1][j-weight[i][0]]+value[i][0]; if(t>dp[i][j]) dp[i][j]=t;}
        //          if(j>=weight[i][0]+weight[i][1]){t=dp[i-1][j-weight[i][0]-weight[i][1]]+value[i][0]+value[i][1]; if(t>dp[i][j]) dp[i][j]=t;}
        //          if(j>=weight[i][0]+weight[i][2]){t=dp[i-1][j-weight[i][0]-weight[i][2]]+value[i][0]+value[i][2]; if(t>dp[i][j]) dp[i][j]=t;}
        //          if(j>=weight[i][0]+weight[i][1]+weight[i][2]){t=dp[i-1][j-weight[i][0]-weight[i][1]-weight[i][2]]+value[i][0]+value[i][1]+value[i][2]; if(t>dp[i][j]) dp[i][j]=t;}

    }

    cout << dp[m][N]*10 <<endl;
    return 0;
}

```

17 坐标移动

开发一个坐标计算工具，A表示向左移动，D表示向右移动，W表示向上移动，S表示向下移动。从(0,0)点开始移动，从输入字符串里面读取一些坐标，并将最终输入结果输出到输出文件里面。

输入：合法坐标为A(或者D或者W或者S) + 数字（两位以内）坐标之间以;分隔。非法坐标点需要进行丢弃。如AA10; A1A; \$\$\$; YAD; 等。

输入

复制

A10;S20;W10;D30;X;A1A;B10A11;;A10;

输出

复制

10,-10

```

#include<iostream>
#include<vector>
#include<string>
using namespace std;

```

```

int move(string str){
    int x=0, y=0; //初始纵横坐标
    int len = str.size(); ///字符串长度
    vector <string> vec; ///存分割后的子字符串
    int sublen = 0; //每个子字符串长度
    for(int i=0; i<len ; i++){
        if(str[i]!=';'){sublen++;continue;}
        vec.push_back(str.substr(i-sublen,sublen));
        sublen = 0;
    }
    for(int i=0; i<vec.size(); i++){ ///每个字符串循环
        int num = 0;
        if((vec[i].size()==3)&& vec[i][1]>='0' && vec[i][1]<='9' && vec[i]
[2]>='0' && vec[i][2]<='9' )
        { num = (vec[i][1]-'0')*10 + (vec[i][2]-'0');}
        if(vec[i].size()==2 && vec[i][1]>='0' && vec[i][1]<='9')
        { num = vec[i][1]-'0';}
        if(vec[i].size()==1){num = 0;}

        switch(vec[i][0]){
            case 'A': x-=num;break;
            case 'D': x+=num;break;
            case 'W': y+=num;break;
            case 'S': y-=num;break;
            default: break;
        }
    }
    cout << x << ',' << y << endl;
    return 0;
}

int main(){
    string str;
    cin >> str;
    move(str);
    return 0;
}

```

18 解析IP地址和对应的掩码

请解析IP地址和对应的掩码，进行分类识别。要求按照A/B/C/D/E类地址归类，不合法的地址和掩码单独归类。

所有的IP地址划分为 A,B,C,D,E五类

A类地址1.0.0.0~126.255.255.255;

B类地址128.0.0.0~191.255.255.255;

C类地址192.0.0.0~223.255.255.255;

D类地址224.0.0.0~239.255.255.255;

E类地址240.0.0.0~255.255.255.255

私网IP范围是:

10.0.0.0 ~ 10.255.255.255

172.16.0.0 ~ 172.31.255.255

192.168.0.0 ~ 192.168.255.255

子网掩码为二进制下前面是连续的1，然后全是0。（例如：255.255.255.32就是一个非法的掩码）

注意二进制下全是1或者全是0均为非法

注意:

1. 类似于【0.***】和【127.***】的IP地址不属于上述输入的任何一类，也不属于不合法ip地址，计数时可以忽略

2. 私有IP地址和A,B,C,D,E类地址是不冲突的

输入

复制

10.70.44.68~255.254.255.0

1.0.0.1~255.0.0.0

192.168.0.2~255.255.255.0

19..0.~255.255.255.0

输出

复制

1 0 1 0 0 2 1

思路

1. 按行读取输入，根据字符 '~' 将IP地址与子网掩码分开
2. 查看子网掩码是否合法。
 - 合法，则继续检查IP地址
 - 非法，则相应统计项+1，继续下一行的读入
3. 查看IP地址是否合法
 - 合法，查看IP地址属于哪一类，是否是私有ip地址；相应统计项+1
 - 非法，相应统计项+1

具体实现

1. 判断IP地址是否合法，如果满足下列条件之一即为非法地址
 - 数字段数不为4
 - 存在空段，即【192..1.0】这种
 - 某个段的数字大于255
2. 判断子网掩码是否合法，如果满足下列条件之一即为非法掩码
 - 不是一个合格的IP地址
 - 在二进制下，不满足前面连续是1，然后全是0
 - 在二进制下，全为0或全为1
3. 如何判断一个掩码地址是不是满足前面连续是1，然后全是0？
 - 将掩码地址转换为32位无符号整型，假设这个数为b。如果此时b为0，则为非法掩码
 - 将b按位取反后+1。如果此时b为1，则b原来是二进制全1，非法掩码
 - 如果b和b-1做按位与运算后为0，则说明是合法掩码，否则为非法掩码

```
#include<iostream>
#include<sstream>
#include<string>
#include<vector>
using namespace std;

bool judge_ip(string ip){
    istringstream iss(ip);
    string seg;
    int j = 0;
    while (getline(iss,seg,'.')){
        if(++j>4 || seg.empty() || stoi(seg)>255){return false;}
    }
    return true;
}

bool judge_private(string ip){
    istringstream iss(ip);
    string seg;
    vector<int> v;
    while(getline(iss,seg,'.'))
        v.push_back(stoi(seg));
    if(v[0]==10) return true;
    if(v[0]==172 && v[1]>=16 && v[1]<=31) return true;
    if(v[0]==192 && v[1]== 168) return true;
    return false;
}

bool judge_mask(string ip){
    istringstream iss(ip);
    string seg;
```

```

    unsigned b=0;
    while(getline(iss,seg,','))
        b = (b<<8)+stoi(seg);
    if(!b) return false;
    b = ~b + 1;
    if(b==1) return false;
    if((b&(b-1))==0) return true;
    return false;
}

int main(){
    string input;
    int a=0, b=0, c=0, d=0, e=0, err=0, p=0;
    while(cin >> input){
        istringstream iss(input);
        string add;
        vector<string> v;
        while(getline(iss,add,','))
            v.push_back(add);
        if(!judge_ip(v[1]) || !judge_mask(v[1])) err++;
        else if(!judge_ip(v[1])) err++;
        else{
            int first = stoi(v[0].substr(0,v[0].find_first_of(',')));
            if(judge_private(v[0])) p++;
            if(first>0 && first<127) a++;
            else if(first>127 && first<192) b++;
            else if(first>191 && first<224) c++;
            else if(first>223 && first<240) d++;
            else if(first>239 && first<256) e++;
        }
    }
    cout<<a<<" "<<b<<" "<<c<<" "<<d<<" "<<e<<" "<<err<<" "<<p<<endl;
    return 0;
}

```

19 简单错误记录

题目描述

开发一个简单错误记录功能小模块，能够记录出错的代码所在的文件名称和行号。

处理：

- 1、记录最多8条错误记录，循环记录，最后只用输出最后出现的八条错误记录。对相同的错误记录只记录一条，但是错误计数增加。最后一个斜杠后面的带后缀名的部分（保留最后16位）和行号完全匹配的记录才做算是“相同”的错误记录。
- 2、超过16个字符的文件名称，只记录文件的最后有效16个字符；
- 3、输入的文件可能带路径，记录文件名称不能带路径。
- 4、循环记录时，只以第一次出现的顺序为准，后面重复的不会更新它的出现时间，仍以第一次为准

输入：

```
D:\zwtymj\xccb\ljj\cqzlyaszjvlsjmkwoqijggmybr 645
E:\je\rzuwnjvnuz 633
C:\km\tgjwpb\gy\atl 637
F:\weioj\hadd\connsh\rwyfvzsopsuiqjnr 647
E:\ns\mfwj\wqkoki\eez 648
D:\cfmwafhhgeyawnoo1 649
E:\czt\opwip\osn11\c 637
G:\nt\f 633
F:\fop\yzwqaop 631
F:\yay\jc\yzwqaop 631
```

输出:

```
rzuwnjvnuz 633 1
atl 637 1
rwyfvzsopsuiqjnr 647 1
eez 648 1
fmwafhhgeyawnoo1 649 1
c 637 1
f 633 1
yzwqaop 631 2
```

```
#include<iostream>
#include<vector>
#include<map>

using namespace std;

string decomp(string filepath){
    int end_pos = filepath.size()-1;
    for(; end_pos >= 0; end_pos--){
        if(filepath[end_pos] == '\\') break;
    }
    string file = filepath.substr(end_pos + 1, filepath.size() - end_pos - 1);
    if(file.size() > 16) file = file.substr(file.size()-16, 16);
    return file;
}

int main(){
    string filepath, length;
    map<string,int> m;        ///记录次数
    int start_index = 0;
    vector<string> record(8,""); ///记录串
    while(cin>>filepath>>length){
        string file = decomp(filepath);
        string key = file+" "+length;
        if(m.find(key)==m.end()){///判断表里有没有
            m[key] = 1;///记录重复次数
            record[start_index] = key;
            start_index = (start_index+1)%8; ///1...8循环替换
        }
        else{m[key]+=1;}
    }

    for(int i = 0; i < 8; i++){
        if(record[start_index] != ""){
```

```

        cout << record[start_index] << " " << m[record[start_index]] << endl;
    }
    start_index = (start_index + 1) % 8; ///下一个数开始 很巧妙
}
return 0;
}

```

20 密码验证程序

1.长度超过8位; 2.包括大小写字母.数字.其它符号,以上四种至少三种; 3.不能有相同长度大于2的子串重复

```

#include<iostream>
using namespace std;
//密码合格验证的函数接口
string Password_Verification(string str)
{
    int len = str.size(); //获取字符串长度
    int a[4] = {0}; //初始化一个数组,用 0 1 记录每一种类型字符出现与否的情况
    int count = 0; //计数器,记录字符串包含多少种字符(大小写字母,数字,其他符号等)
    int num = 0; //有相同长度超 2 的子串重复的字符串对数
    //记录字符串是否包含四种字符的情况
    for (int i = 0; i < len; i++)
    {
        if ((str[i] >= 'A') && (str[i] <= 'Z'))
        {
            a[0] = 1;
        }
        else if ((str[i] >= 'a') && (str[i] <= 'z'))
        {
            a[1] = 1;
        }
        else if ((str[i] >= '0') && (str[i] <= '9'))
        {
            a[2] = 1;
        }
        else
        {
            a[3] = 1;
        }
    }
    //计算字符串包含的字符种类总数
    for (int i = 0; i < 4; i++)
    {
        if (a[i] == 1)
        {
            count++;
        }
    }
    //判断字符串是否包含长度超 2 的两个以上相同子串,故考虑长度为 3 的子字符串是否有重复即可,
    从而子字符串有 (len - 2) 种可能,但作为基准子字符串的只需要 (len - 3) 个即可
    //因为最后一个子字符串被反复判断的次数最多,并且在它之后没有可以比较的子字符串了,所以 i <=
    len - 6,即最后一个基准子字符串是倒数第二个子字符串
    for (int i = 0; i <= len - 6; i++)
    {
        for (int j = i + 3; j <= len - 3; j++)

```

```

    {
        if ((str[i] == str[j]) && (str[i + 1] == str[j + 1]) && (str[i + 2]
== str[j + 2]))
        {
            num++;
        }
        //一旦出现重复的子字符串（长度为 3 ），就及时跳出 for 循环，避免做无用功
        if (num != 0)
        {
            break;
        }
    }
    //一旦出现重复的子字符串（长度为 3 ），就及时跳出 for 循环，避免做无用功
    if (num != 0)
    {
        break;
    }
}
//验证密码是否合格，并输出验证结果
if ((len >= 9) && (count >= 3) && (num == 0))
{
    return "OK";
}
//验证不通过的条件也可以放在不同位置，比如先判断字符串的长度是否超过 8，若不满足，其他条件
也就不需要验证了。
else if ((len <= 8) || (count <= 2) || (num >= 1))
{
    return "NG";
}
}

//主函数
int main()
{
    string str;
    while (getline(cin, str))
    {
        cout << Password_Verification(str) << endl;
    }
    return 0;
}

```

28 素数伴侣

若两个正整数的和为素数，则这两个正整数称之为“素数伴侣”，如2和5、6和13，它们能应用于通信加密。现在密码学会请你设计一个程序，从已有的N（N为偶数）个正整数中挑选出若干对组成“素数伴侣”，挑选方案多种多样，例如有4个正整数：2，5，6，13，如果将5和6分为一组中只能得到一组“素数伴侣”，而将2和5、6和13编组将得到两组“素数伴侣”，能组成“素数伴侣”最多的方案称为“最佳方案”，当然密码学会希望你寻找出“最佳方案”。

输入：有一个正偶数N（N≤100），表示待挑选的自然数的个数。后面给出具体的数字，范围为[2,30000]。

输出：输出一个整数K，表示你求得的“最佳方案”组成“素数伴侣”的对数。


```

#include<vector>
#include<list>
#include<iostream>
using namespace std;

class HungarianAlgorithm
{
public:
    typedef std::list< std::pair<size_t, size_t> > pairmatch;
private:
    bool** graph;
    size_t* match;
    bool* request;
    bool dfs(size_t i, size_t ny)
    {
        for (size_t j = 0; j < ny; ++j)
            if (graph[i][j] && request[j])
            {
                request[j] = false;
                if (match[j] == -1 || dfs(match[j], ny))
                {
                    match[j] = i;
                    return request[j] = true;
                }
            }
        return false;
    }
protected:
    pairmatch pairlist;
public:
    template<class type>
    HungarianAlgorithm(type const & G, size_t nx, size_t ny)
    {
        if (nx && ny)
        {
            graph = new bool*[nx];
            for (size_t i = 0; i < nx; ++i)
                graph[i] = new bool[ny];
            match = new size_t[ny];
            request = new bool[ny];

            for (size_t i = 0; i < nx; ++i)
                for (size_t j = 0; j < ny; ++j)
                    graph[i][j] = G(i, j);

            for (size_t j = 0; j < ny; ++j)
                match[j] = -1;

            for (size_t j = 0; j < ny; ++j)
                request[j] = true;

            for (size_t i = 0; i < nx; ++i)
                dfs(i, ny);

            for (size_t j = 0; j < ny; ++j)
                if (match[j] != -1)
                    pairlist.emplace_back(match[j], j);
        }
    }
};

```

```

        for (size_t i = 0; i < nx; ++i)
            delete[] graph[i];
        delete[] graph;
        delete[] match;
        delete[] request;
    }
}

pairmatch const& getmatch()const { return pairlist; }
};

bool isPrime(size_t n)
{
    for (size_t i = 2; i*i <= n; ++i)
        if (n%i == 0)
            return false;
    return true;
}

int main(void)
{
    size_t n;
    size_t data;
    while (cin >> n)
    {
        vector< size_t > X, Y;
        X.reserve(n);
        Y.reserve(n);
        size_t nx = 0;
        size_t ny = 0;
        for (size_t i = 0; i < n; ++i)
        {
            cin >> data;
            if (data % 2)
                X[nx++] = data;
            else
                Y[ny++] = data;
        }

        auto G = [&](size_t i, size_t j) {return isPrime(X[i] + Y[j]); };

        auto p = HungarianAlgorithm(G, nx, ny).getmatch();

        cout << p.size() << endl;
    }
    return 0;
}

```

30 字符串合并

按照指定规则对输入的字符串进行处理。

详细描述：将输入的两个字符串合并。对合并后的字符串进行排序，要求为：下标为奇数的字符和下标为偶数的字符分别从小到大排序。这里的下标意思是字符在字符串中的位置。对排序后的字符串进行操作，如果字符为'0'——'9'或者'A'——'F'或者'a'——'f'，则对他们所代表的16进制的数进行BIT倒序的操作，并转换为相应的大写字符。如字符为'4'，为0100b，则翻转后为0010b，也就是2。转换后的字符

为'2'；如字符为'7'，为0111b，则翻转后为1110b，也就是e。转换后的字符为大写'E'。

举例：输入str1为"dec"，str2为"fab"，合并为"decfab"，分别对"dca"和"efb"进行排序，排序后为"abcdF"，转换后为"5D37BF"

```
#include <iostream>
#include <algorithm>
using namespace std;
//字符串合并处理的函数接口
void Process_String(string str1, string str2, string strOutput)
{
    //字典法：只考虑 '0' 到 '9' ，'a' 到 'f'，'A' 到 'F' 的字符即可，其余字符不做改变，照原输出
    char Input[] = {"0123456789abcdefABCDEF"}; //输入参照字典（数字 + 大小写字母）
    // int Output[] = "084c2a6e195d3b7f5d3b7f"; //输出参照字典（小写）
    char output[] = {"084C2A6E195D3B7F5D3B7F"}; //输出参照字典（数字 + 大写字母）
    strOutput = str1 + str2; //合并两个字符串
    string odd_str; //下标为奇数的字符组成的字符串，奇数位字符串
    string even_str; //下标为偶数的字符串组成的字符串，偶数位字符串
    //根据字符在合并字符串中的次序，按字典序分奇数位、偶数位独立来排，但次序的奇偶性不变，即原来是奇数位，排序后还是奇数位
    for (int i = 0; i < strOutput.size(); i++)
    {
        if (i % 2 == 0)
        {
            odd_str += strOutput[i];
        }
        else if (i % 2 == 1)
        {
            even_str += strOutput[i];
        }
    }
    sort(odd_str.begin(), odd_str.end()); //奇排序
    sort(even_str.begin(), even_str.end()); //偶排序
    //将按奇数位、偶数位排序后的字符再填回合并字符串 strOutput
    int j = 0; //奇数位字符串的下标
    int k = 0; //偶数位字符串的下标
    for (int i = 0; i < strOutput.size(); i++)
    {
        if (i % 2 == 0)
        {
            strOutput[i] = odd_str[j];
            j++;
        }
        else if (i % 2 == 1)
        {
            strOutput[i] = even_str[k];
            k++;
        }
    }
    //对字符（符合字典 Input[]）所代表的 16 进制的数进行 BIT 倒序的操作，并转换为相应的大写字符
    for (int i = 0; i < strOutput.size(); i++)
    {
        if ((strOutput[i] >= '0') && (strOutput[i] <= '9'))
        {
            strOutput[i] = output[strOutput[i] - '0'];
        }
    }
}
```

```

    }
    else if ((strOutput[i] >= 'a') && (strOutput[i] <= 'f'))
    {
        strOutput[i] = Output[strOutput[i] - 'a' + 10];
    }
    else if ((strOutput[i] >= 'A') && (strOutput[i] <= 'F'))
    {
        strOutput[i] = Output[strOutput[i] - 'A' + 16];
    }
}
cout << strOutput << endl;
return;
}
//主函数
int main()
{
    string str1, str2, strOutput;
    while (cin >> str1 >> str2)
    {
        Process_String(str1, str2, strOutput);
    }
    return 0;
}

```

35 蛇形矩阵

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。

例如，当输入5时，应该输出的三角形为：

1 3 6 10 15

2 5 9 14

4 8 13

7 12

11

```

#include <iostream>
using namespace std;
int main()
{
    int N;
    while(cin>>N)
    {
        for (size_t i = 1; i <= N; i++) //行
        {
            for (size_t j = 1; j < N-(i-1); j++) //列
                cout<< ((j+i-1)*(j+i-1)+j+i-1)/2 - (i-1)<<" ";
            cout<< (N*N+N)/2-(i-1) <<endl; ///避免最后一个有“ ”
        }
    }

    return 0;
}

```

```

//解题思路：笨方法
//蛇形矩阵，顾名思义像蛇摆一样，具有弯曲但连续这个特点；
//从矩阵特点来看，就是沿着方阵的主对角线斜向下方向，以及沿着副对角线斜向上，依次递增 1；
//从输出矩阵元素的个数来看，总数是一个首项为 1 且公差为 1 的等差数列前 N 项和；
//利用等差数列的规律，可以得知，每一行或每一列的数列是一个阶梯级数（相邻两项的差逐渐递增）。
#include <iostream>
using namespace std;
//输出蛇形矩阵的函数接口
int SerpentineMatrix (int N) {
    int Column_d = 1; //初始化列间相邻两项的第一个差
    int ColumnFirstItem = 1; //列首项
    int RowFirstItem = 1; //行首项
    //按行顺序输出矩阵元素
    for (int i = 1; i <= N; i++) {
        RowFirstItem = ColumnFirstItem; //将第 1 列的第 i 个元素按顺序拷贝作为第 i 行的
        首个元素
        cout << ColumnFirstItem << ' '; //输出第 i 行的首个元素
        ColumnFirstItem += Column_d;
        Column_d++; //相邻两项的差递增 1
        int Row_d = i + 1; //初始化第 i 行元素的行间相邻两项的第一个差
        //按列顺序输出第 i 行的矩阵元素
        for (int j = i + 1; j <= N; j++) {
            RowFirstItem += Row_d;
            cout << RowFirstItem << ' '; //输出第 i 行除首个元素之外的其余元素
            Row_d++; //相邻两项的差递增 1
        }
        cout << endl; //第 i 行元素输出结束，换行。
    }
    return 0;
}
//主函数
int main () {
    int N; //矩阵的阶数 N，由于是正整数，故大于或等于 1。
    while (cin >> N) {
        SerpentineMatrix (N);
    }
    return 0;
}

```

107 求解立方根

```

#include <stdio.h>
inline double abs(double x){return (x>0?x:-x);}
double cubert(const double y){
    double x;
    for(x=1.0;abs(x*x*x-y)>1e-7;x=(2*x+y/x/x)/3);
    return x;
}
int main(){
    for(double y;~scanf("%lf",&y);printf("%.1lf\n",cubert(y)));
    return 0;
}

```

