

Week2 感受:

- 1.调试代码时用到的 `system("pause")` 不能带着一起在 oj 提交, 否则会因为无法正常结束程序出现 RE (坑了我三天)。
- 2.要注意题目要求 (输出格式, 空格, 空行, 换行);
- 3.有时候 C 语言里面的函数可能比 C++ 更好用 (比如实时排名问题中使用 `sscanf` 函数读括号内罚时);
- 4.要灵活使用 STL 函数, 大部分 STL 库函数解决问题的效率很高;
- 5.时间和空间不可兼得;
- 6.定义变量尽量用全局变量;
- 7.一个方法不行就换个方法, 出问题一定是自己的错;

## A – Maze

<https://vjudge.net/contest/359331#status/MLongyu/A/0/>

东东有一张地图, 想通过地图找到妹纸。地图显示, 0 表示可以走, 1 表示不可以走, 左上角是入口, 右下角是妹纸, 这两个位置保证为 0。既然已经知道了地图, 那么东东找到妹纸就不难了, 请你编一个程序, 写出东东找到妹纸的最短路线。

### Input

输入是一个  $5 \times 5$  的二维数组, 仅由 0、1 两数字组成, 表示法阵地图。

### Output

输出若干行, 表示从左上角到右下角的最短路径依次经过的坐标, 格式如样例所示。数据保证有唯一解。

### Sample Input

```
0 1 0 0 0
0 1 0 1 0
0 1 0 1 0
0 0 0 1 0
0 1 0 1 0
```

### Sample Output

```
(0, 0)
(1, 0)
(2, 0)
(3, 0)
(3, 1)
(3, 2)
(2, 2)
(1, 2)
(0, 2)
```

(0, 3)  
(0, 4)  
(1, 4)  
(2, 4)  
(3, 4)  
(4, 4)

#### Hint

坐标(x, y)表示第 x 行第 y 列，行、列的编号从 0 开始，且以左上角为原点。

另外注意，输出中分隔坐标的逗号后面应当有一个空格。

#### 解题思路：

迷宫问题可以理解成求最短路径问题，可以使用广度优先搜索方法（BFS）+队列来实现。题目中矩阵以 0/1 矩阵方式储存，可以设定四个方向的向量表示四个方向。

```
int dx[4] = {-1, 1, 0, 0};  
int dy[4] = {0, 0, -1, 1};
```

每次移动进行判断以保证不会超出矩阵范围且不会走到已经走过的点，没确定一个点就把它存入路径数组并且标记该点防止重复经过同一点。BFS 过程中如果发现到达了目标点就利用已经保存的路径进行回溯。经过点的存储结构为

```
struct point {  
    int steps;//fangxiang  
    int p[30];  
    int x, y;  
};
```

其中 steps 为到达当前点需要的步数，数组 p 为到达当前点需要经过哪些步骤：

```
if (u.x == endX && u.y == endY) {/*arrive at the destiny*/  
    int xx = 0, yy = 0;  
    cout << "(" << xx << ", " << yy << ")\n";  
    for (int i = 1; i <= u.steps; i++) {  
        xx = xx + dx[u.p[i]];  
        yy = yy + dy[u.p[i]];  
        cout << "(" << xx << ", " << yy << ")\n";  
    }  
    return;  
}
```

#### 完整代码：

```
#include <cstring>  
#include <iostream>  
using namespace std;  
  
int g[6][6];  
bool visited[6][6];  
struct point {  
    int steps;//fangxiang  
    int p[30];  
    int x, y;  
};
```

```

int startX = 0, startY = 0, endX = 4, endY = 4;
int dx[4] = {-1, 1, 0, 0};
int dy[4] = {0, 0, -1, 1};
point que[30], v, u, s;

void bfsPath() {
    s.x = startX;
    s.y = startY;
    s.steps = 0;
    s.p[s.steps] = 0;
    int f = 0, e = 0;
    que[e++] = s;
    visited[startX][startY] = true;
    while (f <= e) {
        u = que[f++];
        if (u.x == endX && u.y == endY) { /*arrive at the destiny*/
            int xx = 0, yy = 0;
            cout << "(" << xx << ", " << yy << ")\n";
            for (int i = 1; i <= u.steps; i++) {
                xx = xx + dx[u.p[i]];
                yy = yy + dy[u.p[i]];
                cout << "(" << xx << ", " << yy << ")\n";
            }
            return;
        }
        for (int i = 0; i < 4; ++i) {
            int nx = u.x + dx[i];
            int ny = u.y + dy[i];
            if (nx >= 0 && nx < 5 && ny >= 0 && ny < 5 && g[nx][ny] != 1 &&
                !visited[nx][ny]) { /*不出界且未走过*/
                v.x = nx;
                v.y = ny;
                v.steps = u.steps + 1;
                for (int j = 0; j < u.steps; ++j) {
                    v.p[j] = u.p[j];
                }
                v.p[v.steps] = i;
                que[e++] = v;
                visited[nx][ny] = true;
            }
        }
    }
}

int main() {
    int m = 5, n = 5;
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> g[i][j];

```

```
    }  
}  
memset(visited, false, sizeof(visited));  
bfsPath();  
//system("pause");  
return 0;  
}
```

## B - Pour Water

<https://vjudge.net/contest/359331#problem/B>

倒水问题 "fill A" 表示倒满 A 杯, "empty A"表示倒空 A 杯, "pour A B" 表示把 A 的水倒到 B 杯并且把 B 杯倒满或 A 倒空。

### Input

输入包含多组数据。每组数据输入 A, B, C 数据范围  $0 < A \leq B$ 、 $C \leq B \leq 1000$ 、A 和 B 互质。

### Output

你的程序的输出将由一系列的指令组成。这些输出行将导致任何一个罐子正好包含 C 单位的水。每组数据的最后一行输出应该是"success"。输出行从第 1 列开始, 不应该有空行或任何尾随空格。

### Sample Input

2 7 5

2 7 4

### Sample Output

fill B

pour B A

success

fill A

pour A B

fill A

pour A B

success

### Notes

如果你的输出与 Sample Output 不同, 那没关系。对于某个"A B C"本题的答案是多解的, 不能通过标准的文本对比来判定你程序的正确与否。所以本题由 SPJ (Special Judge) 程序来判定你写的代码是否正确。

### 解题思路:

可以把整个倒水的过程形成得到的图当做一个六叉树(六种操作), 其中部分分支经过判断无法生成, 而我们要做的就是在这颗树中找到某一节点, 此时两个水杯其中有一个水量为目标水量。和先前做过的迷宫思路一样, 也是以最短路径找到目标点的过程, 采用 BFS 广度优先搜索+队列的方法, 从初始状态开始, 逐一判断当前状态可进行的操作并生成相应分支直到其中一杯水满足要求。

在此过程中还要确保不会产生已产生的两个杯子的状态(由 AB[1001][1001]数组辅助实现), opNum[x][y]数组用于储存两个杯子水量分别为 x, y 状态的上一步操作的操作数  
输出步骤时将每一步操作数存在一个数组然后利用它输出操作。

### 完整代码:

```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;
struct status //记录状态的结构体
{
```

```

int A, B; // A, B 杯子水含量
status(int _A, int _B) {
    A = _A;
    B = _B;
}
};
void Print(int& x, int& y, int& B);
int AB[1001][1001], opNum[1001][1001];
void bfs(int A, int B, int C) //广度优先搜索
{
    queue<status> q;
    q.push(status(0, 0));
    for (int i = 0; i <= A; i++) {
        for (int j = 0; j <= B; j++) {
            AB[i][j] = -1;
        }
    }

    AB[0][0] = 0;
    while (!q.empty()) {
        status cur = q.front();
        q.pop();
        int a = cur.A, b = cur.B;
        //switch
        for (int i = 0; i < 6; i++) {
            int x = -1, y = -1;
            if (i == 0 && a < A && -1 == AB[A][b]) {
                x = A;
                y = b;
            }
            if (i == 1 && b < B && -1 == AB[a][B]) {
                x = a;
                y = B;
            }
            if (i == 2 && b < B && a > 0) {
                if (a > B - b && -1 == AB[a - B + b][B]) {
                    x = a - B + b;
                    y = B;
                }
                if (a < B - b && -1 == AB[0][b + a]) {
                    x = 0;
                    y = b + a;
                }
            }
        }
    }
}

```

```

        if (i == 3 && a < A && b > 0) {
            if (b > A - a && -1 == AB[A][b - A + a]) {
                y = b - A + a;
                x = A;
            }
            if (b < A - a && -1 == AB[b + a][0]) {
                y = 0;
                x = b + a;
            }
        }
        if (i == 4 && -1 == AB[0][b]) {
            x = 0;
            y = b;
        }
        if (i == 5 && -1 == AB[a][0]) {
            x = a;
            y = 0;
        }

        if (x != -1 && y != -1) {
            opNum[x][y] = i;
            AB[x][y] = a * B + a + b;
            if (x == C || y == C) //罐子中包含C单位的水，结束搜索
            {
                Print(x, y, B);
                return;
            }
            q.push(status(x, y));
        }
    }
}

void Print(int& x, int& y, int& B) //输出
{
    int a = x, b = y;
    vector<int> ans;
    while (AB[a][b] != a * B + a + b) {
        int ta = AB[a][b] / (B + 1), tb = AB[a][b] % (B + 1);
        ans.push_back(opNum[a][b]);
        a = ta;
        b = tb;
    }
    for (int i = ans.size() - 1; i >= 0; i--) {
        switch (ans[i]) {

```

```

        case 0:
            cout << "fill A\n";
            break;
        case 1:
            cout << "fill B\n";
            break;
        case 2:
            cout << "pour A B\n";
            break;
        case 3:
            cout << "pour B A\n";
            break;
        case 4:
            cout << "empty A\n";
            break;
        default:
            cout << "empty B\n";
            break;
    }
}
cout << "success\n";
}
int main() {
    int A, B, C;
    while (cin >> A >> B >> C)
        if (0 == C)
            cout << "success\n";
        else
            bfs(A, B, C);
    // system("pause");/*吸取教训，这玩意不能一块带进 vj*/
    return 0;
}

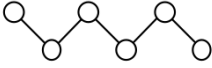
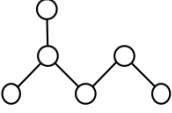
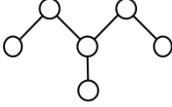
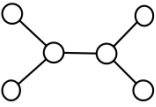
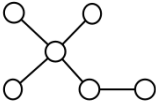
```



## A - 化学 (编译器选 GNU G++)

<https://vjudge.net/contest/359621#problem/A>

化学很神奇，以下是烷烃基。

Name	HDMG	Name	HDMG
n-hexane		2-methylpentane	
3-methylpentane		2,3-dimethylbutane	
2,2-dimethylbutane		-	-

假设如上图，这个烷烃基有 6 个原子和 5 个化学键，6 个原子分别标号 1~6，然后用一对数字 a,b 表示原子 a 和原子 b 间有一个化学键。这样通过 5 行 a,b 可以描述一个烷烃基  
你的任务是甄别烷烃基的类别。

原子没有编号方法，比如

1 2

2 3

3 4

4 5

5 6

和

1 3

2 3

2 4

4 5

5 6

是同一种，本质上就是一条链，编号其实是没有关系的，可以在纸上画画就懂了

### Input

输入第一行为数据的组数  $T$  ( $1 \leq T \leq 200000$ )。每组数据有 5 行，每行是两个整数 a, b ( $1 \leq a, b \leq 6, a \leq b$ )

数据保证，输入的烷烃基是以上 5 种之一

### Output

每组数据，输出一行，代表烷烃基的英文名

### Example

Input

```
2
1 2
2 3
3 4
4 5
5 6
1 4
2 3
3 4
4 5
5 6
```

Output

```
n-hexane
3-methylpentane
```

**解题思路：**

仔细观察五种同分异构体，发现最主要的区别在于原子的杂化方式（相邻连接的原子数）和不同类型原子的个数，其中有三个（n-hexane, 2,2-dimethylbutane, 2,3-dimethylbutane）可以通过判断具有各种分支数点的个数直接判断出来，而剩下两个则需要判断三分支点的所有邻接点中为二分支点个数，以此来区分。

每个烷烃对应一个图，图以邻接矩阵进行存储，确定每个点邻接点个数然后进行后面的判断

**完整代码：**

```
#include "iostream"
using namespace std;
int have3node;
int main() {
    int n, a, b;
    cin >> n;
    for (int i = 0; i < n; i++) {

        int num[6] = {0}; //存每个节点有几个连接的点
        int numOf1 = 0, numOf2 = 0, numOf3 = 0,
            numOf4 = 0; //比如 numOf3 就是有三个相邻点的节点数
        bool g[6][6]; //邻接矩阵
        //初始化图
        for (int j = 0; j < 6; j++)
            for (int k = 0; k < 6; k++) g[j][k] = false;
        //连接节点
        for (int u = 0; u < 5; u++) {
            cin >> a >> b;
            g[a - 1][b - 1] = true;
            g[b - 1][a - 1] = true;
        }
    }
}
```

```

    }
    for (int x = 0; x < 6; x++) {
        for (int y = 0; y < 6; y++)
            if (g[x][y]) num[x]++;
        switch (num[x]) {
            case 1:
                numOf1++;
                break;
            case 2:
                numOf2++;
                break;
            case 3:
                numOf3++;
                have3node = x;
                break;
            case 4:
                numOf4++;
                break;
            default:
                break;
        }
    }
    int count = 0;
    if (numOf3 == 1) {
        for (int m = 0; m < 6; m++) {
            if (g[have3node][m]) {
                if (num[m] == 2) count++;
            }
        }
    }
    if (numOf4 == 1)
        cout << "2,2-dimethylbutane\n";
    else if (numOf3 == 2)
        cout << "2,3-dimethylbutane\n";
    else if (numOf1 == 2)
        cout << "n-hexane\n";
    else if (count == 2)
        cout << "3-methylpentane\n";
    else
        cout << "2-methylpentane\n";
}
system("pause");
return 0;
}

```

## B - 爆零(×)大力出奇迹(√)

<https://vjudge.net/contest/359621#problem/B>

程序设计思维作业和实验使用的实时评测系统，具有及时获得成绩排名的特点，那它的功能是怎么实现的呢？

我们千辛万苦熬完了不忍直视的程序并提交以后，评测系统要么返回 AC，要么是返回各种其他的错误，不论是怎样的错法，它总会给你记上一笔，表明你曾经在这儿被坑过，而当你历经千辛将它 AC 之后，它便会和你算笔总账，表明这题共错误提交了几次。

在岁月的长河中，你通过的题数虽然越来越多，但通过每题时你所共花去的时间（从最开始算起，直至通过题目时的这段时间）都会被记录下来，作为你曾经奋斗的痕迹。特别的，对于你通过的题目，你曾经的关于这题的每次错误提交都会被算上一定的单位时间罚时，这样一来，你在做出的题数上，可能领先别人很多，但是在做出同样题数的人中，你可能会因为罚时过高而处于排名上的劣势。

例如某次考试一共八道题 (A,B,C,D,E,F,G,H)，每个人做的题都在对应的题号下有个数量标记，负数表示该学生在该题上有过的错误提交次数但到现在还没有 AC，正数表示 AC 所耗的时间，如果正数 a 跟上了一对括号，里面有个正数 b，则表示该学生 AC 了这道题，耗去了时间 a，同时曾经错误提交了 b 次。例子可见下方的样例输入与输出部分。

### Input

输入数据包含多行，第一行是共有的题数 n ( $1 \leq n \leq 12$ ) 以及单位罚时 m ( $10 \leq m \leq 20$ )，之后的每行数据描述一个学生的信息，首先是学生的用户名（不多于 10 个字符的字符串）其次是所有 n 道题的得分现状，其描述采用问题描述中的数量标记的格式，见上面的表格。

### Output

根据这些学生的得分现状，输出一个实时排名。实时排名显然先按 AC 题数的多少排，多的在前，再按时间分的多少排，少的在前，如果凑巧前两者都相等，则按名字的字典序排，小的在前。每个学生占一行，输出名字（10 个字符宽），做出的题数（2 个字符宽，右对齐）和时间分（4 个字符宽，右对齐）。名字、题数和时间分相互之间有一个空格。数据保证可按要求的输出格式进行输出。

### Sample Input

```
8 20
GuGuDong 96 -3 40(3) 0 0 1 -8 0
hrz 107 67 -3 0 0 82 0 0
TT 120(3) 30 10(1) -3 0 47 21(2) -2
OMRailgun 0 -99 -8 0 -666 -10086 0 -9999996
yjq -2 37(2) 13 -1 0 113(2) 79(1) -1
Zjm 0 0 57(5) 0 0 99(3) -7 0
```

### Sample Output

```
TT 5 348
```

yjq	4	342
GuGuDong	3	197
hrz	3	256
Zjm	2	316
OMRailgun	0	0
OMRailgun	0	0

### 解题思路：

按照每个人的答题状况确定 AC 数和分数，并存储在数组中，然后排序输出，这里再次复习了格式化输出。

读取带有罚时的分数是可以用 `res = sscanf(str, "%d(%d)", &h, &t);` 来实现，使用 `qsort` 函数时，作为参数的函数 `cmp()` 的参数必须是 `const` 型的

<https://zh.cppreference.com/w/cpp/algorithm/qsort>

### std::qsort

---

```

    定义于头文件 <cstdlib>
void qsort( void *ptr, std::size_t count, std::size_t size, /*compare-pred*/ comp );      (1)
void qsort( void *ptr, std::size_t count, std::size_t size, /*c-compare-pred*/ comp );
extern "C++" using /*compare-pred*/ = int(const void*, const void*); // 仅为说明      (2)
extern "C" using /*c-compare-pred*/ = int(const void*, const void*); // 仅为说明

```

---

以升序排序 `ptr` 所指向的给定数组。数组含 `count` 个 `size` 字节大小的元素。用 `comp` 所指向的函数比较对象。

若 `comp` 指示二个元素等价，则其顺序未指定。

#### 参数

**ptr** - 指向要排序的数组的指针  
**count** - 数组元素数  
**size** - 数组中元素的大小，以字节表示  
**comp** - 比较函数。若首个参数小于第二个，则返回负整数值，若首个参数大于第二个，则返回正整数值，若两参数相等，则返回零。  
 比较函数的签名应等价于如下形式：

---

```
int cmp(const void *a, const void *b);
```

---

该函数必须不修改传递给它的对象，而且在调用比较相同对象时必须返回一致的结果，无关乎它们在数组中的位置。

### 完整代码：

```

#include <iostream>
using namespace std;
struct student {
    char name[15];
    int AC_Count;
    int score;
};
/*https://zh.cppreference.com/w/cpp/algorithm/qsort*/
/*必须是 const 型参数*/
int cmp(const void *a, const void *b) {
    struct student *aa = (student *)a;
    struct student *bb = (student *)b;
    if (aa->AC_Count != bb->AC_Count)
        return bb->AC_Count - aa->AC_Count;
    else if (aa->score != bb->score)
        return aa->score - bb->score;
}

```

```

        else
            return aa->name - bb->name;
    }
    int n, m;
    student stu[10001];
    int main() {
        int j, t, h, res, num = 0;
        char str[20];
        scanf("%d%d", &n, &m);
        while (scanf("%s", &stu[num].name) != EOF) {
            stu[num].AC_Count = 0;
            stu[num].score = 0;
            for (int i = 0; i < n; i++) {
                scanf("%s", str);
                res = sscanf(str, "%d(%d)", &h, &t);
                if (res == 2) {
                    stu[num].AC_Count++;
                    stu[num].score += h;
                    stu[num].score += m * t;
                } else if (res == 1 && h > 0) {
                    stu[num].AC_Count++;
                    stu[num].score += h;
                }
            }
            num++;
            /*qsort(stu, num, sizeof(stu[0]), cmp);
            for (i = 0; i < num; i++) {
                printf("%-10s %2d %4d\n", stu[i].name, stu[i].AC_Count,
                    stu[i].score);
            }*/
        }
        qsort(stu, num, sizeof(stu[0]), cmp);
        for (int i = 0; i < num; i++) {
            printf("%-
10s %2d %4d\n", stu[i].name, stu[i].AC_Count, stu[i].score);
        }
        return 0;
    }
}

```

# C - 瑞神打牌 （不支持 C++11，G++和 C++编译器都试试）

<https://vjudge.net/contest/359621#problem/C>

瑞神 HRZ 因为疫情在家闲得无聊，同时他又非常厉害，所有的课对他来说都是水一水就能拿 A+，所以他无聊，找来了另外三个人：咕咕东，腾神以及 zjm 来打牌（天下苦瑞神久矣）。显然，牌局由四个人构成，围成一圈。我们称四个方向为北 东 南 西。对应的英文是 North, East, South, West。游戏一共由一副扑克，也就是 52 张构成。开始，我们指定一位发牌员（东南西北中的一个，用英文首字母标识）开始发牌，发牌顺序为顺时针，发牌员第一个不发自己，而是发他的下一个人（顺时针的下一个人）。这样，每个人都会拿到 13 张牌。现在我们定义牌的顺序，首先，花色是（梅花）<（方片）<（黑桃）<（红桃），（输入时，我们用 C,D,S,H 分别表示梅花，方片，黑桃，红桃，即其单词首字母）。对于牌面的值，我们规定  $2 < 3 < 4 < 5 < 6 < 7 < 8 < 9 < T < J < Q < K < A$ 。现在你作为上帝，你要从小到大排序每个人手中的牌，并按照给定格式输出。（具体格式见输出描述和样例输出）。

## Input

输入包含多组数据

每组数据的第一行包含一个大写字母，表示发牌员是谁。如果该字符为 '#' 则表示输入结束。接下来有两行，每行有 52 个字符，表示了 26 张牌，两行加起来一共 52 张牌。每张牌都由两个字符组成，第一个字符表示花色，第二个字符表示数值。

## Output

输出多组数据发牌的结果，每组数据之后需要额外多输出一个空行！！！！

每组数据应该由 24 行的组成，输出按照顺时针方向，始终先输出 South Player 的结果，每位玩家先输出一行即玩家名称（东南西北），接下来五行，第一行和第五行输出固定格式（见样例），第二行和第四行按顺序和格式输出数值（见样例），第三行按顺序和格式输出花色（见样例）。

## Sample Input

```
N
CTCAH8CJD4C6D9SQC7S5HAD2HJH9CKD3H6D6D7H3H4C5DKHKS9
SJDTS3S7S4C4CQHTSAH2D8DJSTSKS2H5D5DQDAH7C9S8C8S6C2C3
#
```

## Sample Output

South player:

```
+---+---+---+---+---+---+---+---+---+---+---+---+
|6 6|A A|6 6|J J|5 5|6 6|7 7|9 9|4 4|5 5|7 7|9 9|T T|
| C | C | D | D | S | S | S | S | H | H | H | H | H |
|6 6|A A|6 6|J J|5 5|6 6|7 7|9 9|4 4|5 5|7 7|9 9|T T|
+---+---+---+---+---+---+---+---+---+---+---+---+
```

West player:

```

+---+---+---+---+---+---+---+---+---+---+
|2 2|5 5|9 9|K K|5 5|7 7|9 9|4 4|T T|J J|A A|8 8|A A|
| C | C | C | C | D | D | D | S | S | S | S | H | H |
|2 2|5 5|9 9|K K|5 5|7 7|9 9|4 4|T T|J J|A A|8 8|A A|
+---+---+---+---+---+---+---+---+---+---+

```

North player:

```

+---+---+---+---+---+---+---+---+---+---+
|3 3|4 4|J J|2 2|3 3|T T|Q Q|K K|8 8|Q Q|K K|2 2|3 3|
| C | C | C | D | D | D | D | D | S | S | S | H | H |
|3 3|4 4|J J|2 2|3 3|T T|Q Q|K K|8 8|Q Q|K K|2 2|3 3|
+---+---+---+---+---+---+---+---+---+---+

```

East player:

```

+---+---+---+---+---+---+---+---+---+---+
|7 7|8 8|T T|Q Q|4 4|8 8|A A|2 2|3 3|6 6|J J|Q Q|K K|
| C | C | C | C | D | D | D | S | S | H | H | H | H |
|7 7|8 8|T T|Q Q|4 4|8 8|A A|2 2|3 3|6 6|J J|Q Q|K K|
+---+---+---+---+---+---+---+---+---+---+

```

#### 解题思路:

定义一个扑克牌的结构体, 包含花色和数值,, 再定义一个 52 张牌的数组表示一副牌每次直接读取一副牌的信息然后按间隔为 4 分发给四个人。

```

struct poke //扑克牌的牌面
{
    char num;
    char color;
};
for (int i = 0; i < 52; i++) {
    cin >> set[i].color >> set[i].num;
}

```

牌的排序部分由 map 完成, 将每张牌牌面数字和花色对应具体大小的数字方便比较。这其中利用了 isdigit()函数和 isalpha()函数来判断数字和字母

```

bool cmp(poke a, poke b) {
    if (a.color == b.color) {
        if ((isdigit(a.num) && isdigit(b.num)) ||
            (isdigit(b.num) && isalpha(a.num)) ||
            (isdigit(a.num) && isalpha(b.num)))
            return a.num < b.num;
        else if (isalpha(a.num) && isalpha(b.num))
            return m[a.num] < m[b.num];
    } else {
        return mm[a.color] < mm[b.color];
    }
}

```



完整代码:

```
#include <algorithm>
#include <iostream>
#include <map>
#include <vector>
using namespace std;

struct poke //扑克牌的牌面
{
    char num;
    char color;
};

//各个人的牌
vector<poke> north;
vector<poke> east;
vector<poke> south;
vector<poke> west;
map<char, int> m;
map<char, int> mm;

bool cmp(poke a, poke b) {
    if (a.color == b.color) {
        if ((isdigit(a.num) && isdigit(b.num)) ||
            (isdigit(a.num) && isalpha(b.num)) ||
            (isdigit(b.num) && isalpha(a.num)))
            return a.num < b.num;
        else if (isalpha(a.num) && isalpha(b.num))
            return m[a.num] < m[b.num];
    } else {
        return mm[a.color] < mm[b.color];
    }
}

poke set[52];
int main() {
    char dir;
    string str1, str2, str;
    m['T'] = 1, m['J'] = 2, m['Q'] = 3, m['K'] = 4, m['A'] = 5;
    mm['C'] = 1, mm['D'] = 2, mm['S'] = 3, mm['H'] = 4;
    while (cin >> dir) {
        if (dir == '#') break;
        for (int i = 0; i < 52; i++) {
            cin >> set[i].color >> set[i].num;
        }
    }
}
```

```

east.clear(), north.clear(), south.clear(), west.clear();
/*
cin >> str1 >> str2;
str = str1 + str2;*/
if (dir == 'N') //如果是 N 发牌，则派牌的顺序是 ESWN 下面的则照顺时针推

```

出

```

{
    for (int i = 0; i < 52; i++) {
        if (i % 4 == 0)
            east.push_back(set[i]);
        else if (i % 4 == 1)
            south.push_back(set[i]);
        else if (i % 4 == 2)
            west.push_back(set[i]);
        else if (i % 4 == 3)
            north.push_back(set[i]);
    }
} else if (dir == 'W') {
    for (int i = 0; i < 52; i++) {
        if (i % 4 == 0)
            north.push_back(set[i]);
        else if (i % 4 == 1)
            east.push_back(set[i]);
        else if (i % 4 == 2)
            south.push_back(set[i]);
        else if (i % 4 == 3)
            west.push_back(set[i]);
    }
} else if (dir == 'E') {
    for (int i = 0; i < 52; i++) {
        if (i % 4 == 0)
            south.push_back(set[i]);
        else if (i % 4 == 1)
            west.push_back(set[i]);
        else if (i % 4 == 2)
            north.push_back(set[i]);
        else if (i % 4 == 3)
            east.push_back(set[i]);
    }
} else if (dir == 'S') {
    for (int i = 0; i < 52; i++) {
        if (i % 4 == 0)
            west.push_back(set[i]);
        else if (i % 4 == 1)

```

```
north.push_back(set[i]);
    else if (i % 4 == 2)
        east.push_back(set[i]);
    else if (i % 4 == 3)
        south.push_back(set[i]);
}
}
```

```
sort(south.begin(), south.end(), cmp);
sort(west.begin(), west.end(), cmp);
sort(north.begin(), north.end(), cmp);
sort(east.begin(), east.end(), cmp);
cout << "South player:" << endl;
cout << "+---+---+---+---+---+---+---+---+---+---+---+---+"
+>" << endl;
for (int i = 0; i < 13; i++)
    cout << "|" << south[i].num << " " << south[i].num;
cout << "| " << endl;
for (int i = 0; i < 13; i++) cout << "|" << south[i].color << " ";
cout << "| " << endl;
for (int i = 0; i < 13; i++)
    cout << "|" << south[i].num << " " << south[i].num;
cout << "| " << endl;
cout << "+---+---+---+---+---+---+---+---+---+---+---+---+"
+>" << endl;
cout << "West player:" << endl;
cout << "+---+---+---+---+---+---+---+---+---+---+---+---+"
+>" << endl;
for (int i = 0; i < 13; i++)
    cout << "|" << west[i].num << " " << west[i].num;
cout << "| " << endl;
for (int i = 0; i < 13; i++) cout << "|" << west[i].color << " ";
cout << "| " << endl;
for (int i = 0; i < 13; i++)
    cout << "|" << west[i].num << " " << west[i].num;
cout << "| " << endl;
cout << "+---+---+---+---+---+---+---+---+---+---+---+---+"
+>" << endl;
cout << "North player:" << endl;
cout << "+---+---+---+---+---+---+---+---+---+---+---+---+"
+>" << endl;
for (int i = 0; i < 13; i++)
    cout << "|" << north[i].num << " " << north[i].num;
cout << "| " << endl;
```

```

for (int i = 0; i < 13; i++) cout << "|" << north[i].color << " ";
cout << "|" << endl;
for (int i = 0; i < 13; i++)
    cout << "|" << north[i].num << " " << north[i].num;
cout << "|" << endl;
cout << "+---+---+---+---+---+---+---+---+---+---+---+---+"
+" << endl;
    cout << "East player:" << endl;
    cout << "+---+---+---+---+---+---+---+---+---+---+---+---+"
+" << endl;
        for (int i = 0; i < 13; i++)
            cout << "|" << east[i].num << " " << east[i].num;
        cout << "|" << endl;
        for (int i = 0; i < 13; i++) cout << "|" << east[i].color << " ";
        cout << "|" << endl;
        for (int i = 0; i < 13; i++)
            cout << "|" << east[i].num << " " << east[i].num;
        cout << "|" << endl;
        cout << "+---+---+---+---+---+---+---+---+---+---+---+---+"
+" << endl;
            cout << endl;
        }

// system("pause");
}

/*
N
CTCAH8CJD4C6D9SQC7S5HAD2HJH9CKD3H6D6D7H3HQH4C5DKHKS9
SJDT5S3S7S4C4CQHTSAH2D8DJSTSKS2H5D5DQDAH7C9S8C8S6C2C3
#
*/

```