Assignment 2 函数调用

本次作业共3道题目。每题设置10个测试案例，其中的5个已经提供给你们用于测试。3道题共计10分。

本次作业的每道题目都有部分代码已经给出（在每道题目对应的文件夹中），你不可以修改已经给出的代码，只能在此模板的基础上完成你的代码。

1.比较大小

题目描述

本题中，你需要实现一个函数，接口如下：

```
int cmp(int num1, int num2);
```

函数的功能是比较两个数的“强度”，如果`num1`更强则返回`1`，`num2`更强则返回`-1`，一样强（即**两个数相等**）则返回`0`。

比较规则依次如下（排在前面的规则优先级更高）：

1. `42`比其他所有数都强。
2. 如果一个数是5的倍数但不是10的倍数，那么这个数比不满足这一条件的数更强。
3. 如果`n`能整除`m`，那么`n`比`m`更强。
4. 如果一个数的最小的质因数是7，那么这个数比不满足这一性质的数更强。
5. 如果上述规则都无法比较出强弱，那么越大的数字越强。

输入格式

（本题的输入部分在`main`函数中已经给出）
输入为两个正整数`num1,num2`

输出格式

（本题的输出部分在`main`函数中已经给出）
输出`cmp(num1,num2)`的返回值

输入

```
42 34
```

输出

```
1
```

输入

```
27 15
```

输出

```
-1
```

输入

```
27 27
```

输出

```
0
```

数据范围

`0 < num1,num2 < 10000`

2. 设计比赛

题目描述

一场比赛有n个参赛选手，每个选手有一个号码$x_1,...,x_n$， 这些选手之间有明确的实力强弱顺序（即在每一场比赛中，较强的选手一定胜出，且不会出现平局），设计一系列比赛来找出其中前三名的选手。

本题中，我们提供接口：

```
bool match(int x1, int x2);
```

表示让号码为`x1`和`x2`的选手进行一场比赛，返回`true`则表示前者获胜，否则表示后者胜利。

你需要通过调用`match`函数最终找出实力前三名的选手，并按照从强到弱的顺序依次输出其编号。

输入格式

输入的第一行为一个正整数`n`，表示参赛选手的人数。

第二行为`n`个正整数$x_1,...,x_n$，x_i表示第`i`个的选手的编号。

输出格式

输出三个编号，依次表示第一、第二、第三强的选手编号。

输入

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

输出

```
3 2 5
```

输入

```
10
10 9 6 7 5 3 8 1 2 4
```

输出

```
10 7 6
```

数据范围

对于所有的输入`n`，`3 < n < 1000`

3. 井字棋

题目描述

在本题中，你需要实现一个简单的井字棋游戏，规则如下：

1. 井字棋的棋盘是一个3行3列的棋盘。
2. 玩家1执`x`，玩家2执`o`，由玩家1开始依次在棋盘的格子中填入棋子。
3. 如果有任意一个玩家的棋子在某一行、某一列、或一条对角线上连成3个己方棋子，那么他获胜。

我们首先定义一个井字棋盘面（即棋盘上的状态）的编码方式：

以这样一个盘面为例（其中`-`表示该格为空）

```
- x -
o o -
- - x
```

我们首先将井字棋的9个格子编号为

```
1 2 3
4 5 6
7 8 9
```

接着我们用x_i表示第i格子的状态，分别用`0,1,2`来表示`-,x,o`三种状态，例如$x_1 = 0, x_2 = 1, x_4 = 2$

最后将盘面编码为$\prod_{i=1}^{9} p_i^{x_i}$，其中$p_i$为第$i$个质数。因此这个盘面的编号为$3^1 \times 7^2 \times 11^2 \times 23^1$

通过这种方式我们可以将一个井字棋盘面唯一地编码成一个正整数，同时一个编码也可以还原出一个井字棋盘面。

基于这种编码方式，在本题中你需要实现以下函数接口：

```
long long play(long long code, int player, int x, int y);
void print(long long code);
int check_winner(long long code);
```

其中：

- `code`表示盘面的编码。**注意：所有盘面编码的变量需要为`long long`类型，否则可能会出现整数溢出的情况。**
- `player`取值为`1`或`2`，表示当前下棋的玩家。
- `x,y`依次表示这一步棋所在的行和列。
- `play(code,p,x,y)`返回下完这一步棋之后的盘面编码。
- `print(code)`会打印出`code`所表示的盘面（格式见上方示例）。
- `check_winner(code)`返回一个整数，`0`表示该盘面尚未分出胜负，`1`表示玩家1（即执`X`方）获胜，`2`表示玩家2（即执`O`方）获胜。我们保证在调用这一函数时最多只有一个玩家获胜（即不存在两个玩家均连成三个棋子的情况）。

（提示：你可以实现一些额外的辅助函数来简化你的程序）

输入格式

（本题中，输入均在`main`函数中处理，因此你的代码不需要处理输入）

输入的第一行为一个正整数`n`，表示总共有`n`步棋。

接下来的`n`行中，每行有两个整数`x,y`，表示当前玩家在第`x`行第`y`列下了一步棋。保证不会出现在同一个格子重复下棋的情况。

输出格式

（本题中，`main`函数会完成所有需要的输出（包括调用`print`），因此你只需要正确实现`print`即可保证输出正确）

输入

```
9
2 2
3 3
3 2
1 2
1 3
3 1
2 1
2 3
1 1
```

输出

```
11
```

```
- - -
- X -
- - -
5819
- - -
- X -
- - O
110561
- - -
- X -
- X O
995049
- O -
- X -
- X O
4975245
- O X
- X -
- X O
1437845805
- O X
- X -
O X O
10064920635
- O X
X X -
O X O
1700971587315
- O X
X X O
O X O
3401943174630
X O X
X X O
O X O
Draw.
```

数据范围

`0 < n < 10`

提交格式

你提交的文件结构应该和以下形式**完全一样**:

```
<your student number>.zip
├── 1_comp
|   └── main.cpp
├── 2_match
|   └── main.cpp
└── 3_Tic-Tac-Toe
    ├── game.cpp
    ├── game.h
    └── main.cpp
```