T1：地牢冒险

题目描述

小蓝鲸是一名勇敢的探险者，准备探索一座神秘的地下城。地下城共有若干层，每进入一层就像调用一个函数一样，新的“调用”被压入函数调用栈中；当探险到达地下最深处 maxLevel 时开始返回，相应的函数调用会依次出栈。现在有如下递归函数 exploreDungeon，它能模拟探险者进入和离开地下城各层时的情景，输出当前的探险深度（从1开始）。

```
void exploreDungeon(int n, int d) {
    Printer p(d);
    if (d >= n) {
        return ;
    } else {
        exploreDungeon(n, d + 1);
    }
}
```

你的任务是在 `dungeon.h` 和 `dungeon.cpp` 中实现 Printer 类，使得程序能正确输出探险者进入和离开地下城各层的过程。

输入输出

输入：一个整数 maxLevel (1 ≤ maxLevel ≤ 20)，表示地牢的总层数。

输出：

- 输出一系列信息，描述探险者进入和离开地下城各层的过程。
- 每行前面根据层级添加对应数量的缩进（每一级两个空格），
- 进入一层 X，输出 Entering level X，离开时输出 Leaving level X。

样例

输入：

```
3
```

输出：

```
Entering level 1
  Entering level 2
    Entering level 3
    Leaving level 3
  Leaving level 2
Leaving level 1
```

提示

Q：评测的机制是什么？

A：评测代码会 `#include "dungeon.h"`，并使用其中定义好的 `Printer` 类来创建对象，并通过给出的递归函数 `exploreDungeon` 来测试你的实现是否正确。你无法看到，也不用关心评测代码的具体实现，只需按照题目要求实现好 `Printer` 类即可。

Q：我应该如何在本地测试我的代码？

A：你可以在 `dungeon.cpp` 中添加一个 `main` 函数，补全输入模块，复制 `exploreDungeon` 函数来测试你的实现。（请注意，提交时请删除 `main` 函数，否则会导致编译失败。）

T2：背包系统

背景说明

在很多游戏中，背包系统都承担着重要的功能。一个合格的背包系统，需要合理管理不同类别、不同品质的物品。玩家可以通过打怪、刷本、完成任务等方式获取物品，并在背包中查询、合成、消耗它们。

接下来，你需要自己设计并实现一个简单的背包系统。为了简化问题，我们的背包只需要存储如下数据：

- 物品类别：每个类别表示一种不同的大类物品（编号从 0 开始），类别总数 $n < 20$。
- 品质：共四档，分别为绿色、蓝色、紫色、橙色
- 数量：同类别、同品质的物品只用一个计数来表示。

同一大类，不同品质的物品间有如下合成关系：

- 3 个绿色 ⇒ 1 个蓝色
- 3 个蓝色 ⇒ 1 个紫色
- 3 个紫色 ⇒ 1 个橙色

功能需求

你的背包系统需要支持以下 四个接口。

1. 增加数量

功能：给定某一类别、某一品质，以及增加的数量，将该数量加入背包。

输入：类别编号 `category`，品质编号 `quality`，增加数量 `delta`（正整数）。

输出：无。

说明：此接口用于模拟玩家拾取物品。

2. 查询数量

功能：查询某一类别物品在各品质上的数量。

输入：类别编号 `category`。

输出：四个整数，用空格隔开，分别表示绿色、蓝色、紫色、橙色物品的数量。

说明：用于前端展示背包中的物品情况。

3. 合成请求

功能：尝试满足一个类别的合成需求。

- 输入为该类别四个品质的“需求数量”。
- 系统需要判定：能否利用当前库存以及“3→1”的合成规则，完成这份需求。

输入：五个整数，第一个整数表示物品大类，后面四个整数表示各品质的需求数量。

输出：如果能够完成，输出需要消耗的各品质物品数量，按照绿蓝紫橙的顺序，使用空格隔开；如果不能完成，则输出 `-1 -1 -1 -1`。

核心要求：

- 由于不能越级合成，且每次合成需要额外消耗相同的金币资源，你给出的合成策略应当是所有可行策略中消耗金币资源最少的。
- 不能跨类别合成。

4．消耗物品

功能：对背包内的一批物品进行扣减操作。

输入：一个变长的序列，每个操作包含：类别编号 + 四个品质的扣减数量（五个以空格分隔的整数）；以类别编号 `-1` 作为结束标记。

输出：无。

说明：输入数据保证合法，即不会出现负数库存。

输入输出

输入请求。每个请求先输入一个整数，表示请求类型 $q \in \{-1, 1, 2, 3, 4\}$。然后按照各个请求的格式输入参数。当输入请求为 `-1` 时，代表结束输入。

样例

输入

```
1 0 0 20
1 0 1 2
2 0
1 1 0 5
1 1 1 5
2 1
4 0 1 1 0 0 1 2 1 0 0 -1
2 0
2 1
3 0 1 2 1 0
4 0 13 1 0 0 -1
2 0
3 0 0 0 1 0
1 0 0 3
3 0 0 0 1 0
4 0 9 0 0 0 -1
2 0
-1
```

输出

```
20 2 0 0
5 5 0 0
19 1 0 0
3 4 0 0
13 1 0 0
6 0 0 0
-1 -1 -1 -1
9 0 0 0
0 0 0 0
```

得分说明

- 对于 30% 的数据，保证只有增加数量和查询数量两个需求。
- 对于另外 30% 的数据，保证没有合成请求需求。
- 对于所有数据，物品总数 $n \leq 20$，请求数 $m \leq 200$。
- 对于全部数据，保证消耗操作的合法性，即不会出现负数库存。

提示

Q：如何设计合理的背包系统？

A：需要根据需求设计对应的类，并通过合理使用成员对象和成员函数来实现数据维护和操作。

Q：如何确保合成消耗的金币最少？

A：拿 9 个绿色合成 3 个蓝色，再拿 3 个蓝色合成 1 个紫色，开销会比拿 3 个蓝色直接合成一个紫色要大，因为进行的合成操作更多（$4 > 3$）。

Q：我应该如何组织文件结构？

A：本题目体提供了 `inventory.cpp`、`inventory.h` 和 `main.cpp` 三个文件，请合理利用这三个文件实现多文件编程，并确保 `main` 函数位于 `main.cpp` 中，否则将无法编译通过。

T3：放苹果

题目描述

将 M 个相同的苹果放在 N 个相同的盘子里，允许有的盘子空着不放，请输出放置的所有方案。（注意：如果7个苹果放入3个盘子，5，1，1和1，5，1

的放置是同一种放法）

输入输出

输入：两个整数 `M` 和 `N`，分别表示苹果的数量和盘子的数量。

输出：若干行，每行输出 `N` 个空格隔开的整数，表示一种放置方案（空着的盘子用 `0` 表示）。放置方案按字典序降序排列。

样例

输入

```
7 3
```

输出

```
7 0 0
6 1 0
5 2 0
5 1 1
4 3 0
4 2 1
3 3 1
3 2 2
```

数据范围

对于 `100%` 的数据，$1 \leq M \leq 20, 1 \leq N \leq 20$

提示

Q：如何枚举所有的放置方案？

A：假设我们有一个函数 `f(M, N)` 可以按照字典序降序输出所有将 `M` 个苹果放入 `N` 个盘子的方案。我们将第一个盘子放上 `k(1 ≤ k ≤ M)` 个苹果的方案抽出来，不难发现，去掉最前面的这个 `k`，剩下的部分恰好就是 `f(M-k, N-1)` 的所有方案！以样例为例：

我们抽出 `f(7, 3)` 中第一个盘子放 `4` 个苹果的两个方案：

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

再手动计算一下 `f(3, 2)`：

```
3 0
2 1
```

可以发现，前者去掉前面的 4 后，正好等于后者。基于此性质，我们可以用“递归函数”来实现 `f(M, N)`。