Homework 5

1. 柜台排队系统

题目背景

在柜台排队的时候, 会有多个窗口需要排队, 我们假设每个柜台处理问题的时间一致, 因此每一个后来的人会被分配到人最少且**排队时间最短**的柜台去排队, 如果有两个柜台都是空的, 那么系统会将其指派到较靠前的柜台. 请实现一个程序, 计算出当一个柜台处理一个人需要t分钟当情况下, n个人在m个柜台中排队需要花多久, 以及这n个人分别在那个柜台排队.

输入输出

题目的输入分两行: 第一行输入m, n, t, 代表柜台数, 待排队的人数和柜台处理单个客人的时间; 第二行有n个数字, 代表第i个客人到来的时间.
题目的输出也分两部分: 第一行输出最后一个客人离开的时间; 第二行输出n个数字, 代表每个客人排队的柜台编号.

数据限制: $1 \leq m \leq 10, 1 \leq n \leq 1000, 0 \leq t \leq 100$, 输入均为正整数.

Sample Case

Input:

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

Output:

```
6
0 1 0
```

解释: 由输入得知一共有3个客人去2个柜台排队, 柜台处理一个客人的时间是3分钟. 第一人客人在$t = 0$的时候进入, 排在0号柜台; 第二个客人在$t = 1$的时候进入, 由于第0个柜台已经被占用, 因此他排在空的1号柜台; 第三个客人在$t = 2$的时候进入, 这个时候前两个客人都没有走, 因此他排在时间较短的0号柜台. 最终最后一名客人离开的时间是第6分钟.

Input:

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

Output:

```
8
0 1 0 1
```

解释: 同上个例子可知前两个客人分别排在$0, 1$号柜台; 第三个客人来的时候第一个客人正好离开, 因此他排在1号柜台; 第4个客人来的时候$0, 1$号柜台都只有一个人排队, 但是由于1号柜台排队时间更短, 因此他排在1号柜台. 最终最后一名客人离开的时间是第8分钟, 在第1柜台($2 + 3 + 3 = 8$).

2. 动态优先级队列

题目背景

假设你正在开发一个任务管理系统, 该系统需要支持动态地添加、删除和查找任务. 每个任务都有一个优先级, 系统需要始终按照优先级处理任务. 为了实现这些功能, 你需要使用一个抽象数据结构（ADT）来存储任务信息.

输入输出

输入分为多行, 每行为一个操作:
第一行为操作数n, 表示有n个操作.
接下来n行, 每行代表一个操作:

- **"ADD 任务名称 优先级"**: 添加一个任务, 任务名称和优先级用空格隔开.
- **"DELETE 任务名称"**: 删除一个任务.
- **"EMPTY"**: 返回当前队列是否为空.
- **"GET"**: 返回当前优先级最高的任务名称.
- **"POP"**: 删除当前优先级最高的任务(无需返回).

注意: 任务名称由不超过20个字母组成, 且不含空格. 优先级为整数, 范围为1-10000(10000最高).

对每个操作, 输出相应的结果:
"ADD" 操作: 输出"Task added."表示成功添加.
"DELETE" 操作: 输出"Task deleted."表示成功删除, 若任务不存在, 输出"Task not found."
"EMPTY" 操作: 输出当前队列是否为空, 为空则输出 `true`, 否则输出 `false`.
"GET" 操作: 输出优先级最高的任务名称, 若任务列表为空, 输出"No task available."
"POP" 操作: 删除优先级最高的任务, 若任务列表为空, 输出"No task available."

数据范围:
操作数n不超过10000.
任务名称由不超过20个字母组成, 且不含空格.
优先级为整数, 范围为$1 - 10000$, 其中10000为最高优先级.
任务名称与优先级都不重复.

注意事项

本题中可以使用任何数据结构, 因此建议大家使用 `queue(priority_queue)` 和 `map` 或者其他数据结构完成本题. **任务名称和优先级不重复**.

Sample Case

Input:

```
8
ADD TaskA 3
ADD TaskB 5
ADD TaskC 1
GET
DELETE TaskB
POP
GET
EMPTY
```

Output:

```
Task added.
Task added.
Task added.
TaskB
Task deleted.
TaskC
false
```

3. 黑白棋

题目背景

黑白棋, 又叫翻转棋（Reversi）. 黑白棋在西方和日本很流行. 游戏通过相互翻转对方的棋子, 最后以棋盘上谁的棋子多来判断胜负. 它的游戏规则简单, 黑白棋的棋盘是一个有8×8方格的棋盘. 下棋时将棋下在空格中间, 而不是像围棋一样下在交叉点上. 开始时在棋盘正中有两白两黑四个棋子交叉放置, 黑棋总是先下子. 在下子的时候把自己颜色的棋子放在棋盘的空格上, 而当自己放下的棋子在横、竖、斜八个方向内有一个自己的棋子, 则被夹在中间的全部翻转会成为自己的棋子. 并且, 只有在可以翻转棋子的地方才可以下子.

1．棋局开始时黑棋位于e4和d5, 白棋位于d4和e5.
2．黑方先行, 双方交替下棋.
3．一步合法的棋步包括: 在一个空格新落下一个棋子, 并且翻转对手一个或多个棋子.
4．新落下的棋子与棋盘上已有的同色棋子间, 对方被夹住的所有棋子都要翻转过来. 可以是横着夹, 竖着夹, 或是斜着夹. 夹住的位置上必须全部是对手的棋子, 不能有空格.
5．一步棋可以在数个方向上翻棋, 任何被夹住的棋子都必须被翻转过来, 棋手无权选择不去翻某个棋子.
6．除非至少翻转了对手的一个棋子, 否则就不能落子. 如果一方没有合法棋步, 也就是说不管他下到哪里, 都不能至少翻转对手的一个棋子, 那他这一轮只能弃权, 而由他的对手继续落子直到他有合法棋步可

下.
7．如果一方至少有一步合法棋步可下, 他就必须落子, 不得弃权.
8．棋局持续下去, 直到棋盘填满或者双方都无合法棋步可下.

更多规则请自行了解, 或者自行搜索网站体验.

输入输出

本题要求实现黑白棋中的棋子类和棋盘类.

0. 棋子类
棋子类其实代表一颗棋子的坐标, 包含两个成员x和y, 分别是横坐标与纵坐标. 通过 `Pos pos(int x, int y);` 初始化, 通过 `pos.x`, `pos.y` 获取成员.

1. 初始化棋盘
棋盘有两种初始化方法

```
ChessBoard Board();
ChessBoard Board(vector<vector<int>> board);
```

- 直接初始化, 此时生成一个二维数组, 是初始化的空白棋盘.
- 传入一个8*8的数组, 棋盘直接将该数组作为初始棋盘.

2. 判断棋局是否结束

通过 `bool Board.IsEnd();` 得到棋局是否结束, 如果已经结束(双方都没有合法位置可以下), 那么返回 `true`, 否则返回 `false`.

3. 得到目前的棋盘

通过 `vector<vector<int>> Board.GetBoard();` 得到当前棋盘的二维数组.

4. 得到当前的全部合法位置

通过 `vector<Pos> Board.GetLegalPos(int player);` 得到一个类型为 `Pos` 的向量, 表示从上至下, 从左至右的顺序下每一个位置对于玩家 `player` 来说合法的位置.

5. 获取比分

通过 `vector<int> Board.GetScore();` 得到一个长度为2的数组, 表示两个玩家分别在棋盘上占有多少颗棋子. 例如初始

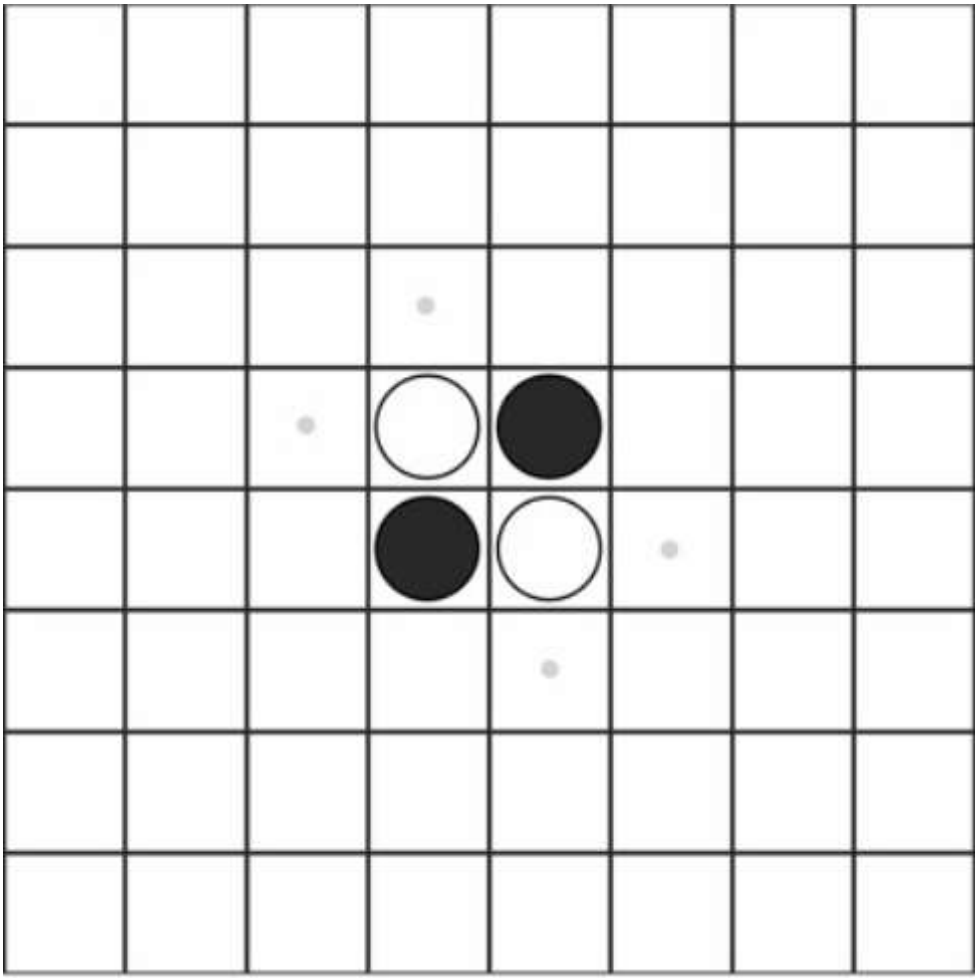
6. 下子

通过 `bool Board.Move(int player, Pos pos);` 表示玩家 `player` 在pos位置上下了一颗子, 返回值表示该行动是否成功. 如果当前位置符合规则, 则落子, 翻转棋子并返回 `true`, 如果当前位置不符合落子要

求, 则返回 `false`.

注意事项

- 玩家使用 `int` 表示, 黑棋先手, 表示为1, 白棋后手, 表示为-1.
- 棋盘统一使用8×8表示, 初始棋盘样式如图所示



- 所有棋盘使用二维的int数组表示, 其中黑子表示为1, 白字表示为-1, 空白表示为0. 同时按照行列顺序排列, 初始的棋盘应该为:

```
[[0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, -1, 1, 0, 0, 0],
 [0, 0, 0, 1, -1, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0]]
```

- `Chess.h` 和 `main.cpp` 已经给出, 请按照要求写 `Chess.cpp`.
- 本题中可以使用任何数据结构, 在不改变接口的情况下可以自行添加函数.

4. Brainfuck 解释器

题目描述
Brainfuck是一种极简主义的编程语言, 由Urban Müller于1993年创建. 它的语法非常简单, 只有8个指令, 包括移动指针、增加或减少指针所指位置的值、输入或输出指针所指位置的值等. 你需要编写一个Brainfuck解释器, 实现对Brainfuck代码的解释执行. 具体地, 你需要编写一个程序, 能够接受一行Brainfuck代码作为输入, 然后输出程序的执行结果.

Brainfuck代码由8个指令组成, 包括:

指令	含义
`>`	将指针向右移动一个位置
`<`	将指针向左移动一个位置
`+`	将指针所指位置的值加1
`-`	将指针所指位置的值减1
`.`	输出指针所指位置的值
`,`	输入一个字符到指针所指位置
`[`	如果指针所指位置的值为0, 跳转到与之匹配的`]`指令之后
`]`	如果指针所指位置的值不为0, 跳转到与之匹配的`[`指令之后

在Brainfuck代码中, 除了以上8个指令以外的所有字符都会被忽略.

输入输出

输入文件包含两行字符串, 表示Brainfuck代码文件的路径, 第二行字符串表示brainfuck程序的输入. 输出程序的执行结果, 对于每个输出指令`.`, 输出指针所指位置的值(Ascii).

Sample Case

Input:

```
++++++++ [->++++++++< ] >+.
```

Output:

```
A
```

解释: 在本样例中, 首先将指针所在的位置(0)的值加1, 重复了8次, 然后进入一个循环. 在这个循环中, 首先将位置(0)的值减1, 接着向右移动一个指针, 将该位置(1)的值加1重复8次. 再向左移动一个指针, 进入循环末尾(`]`), 末尾会判断当前位置的值(0), 如果该值为0则跳出循环, 不为0则继续, 因此这里会循环8次, 将位置(1)的值加到64. 循环结束后, 将指针向右移动一个位置(1)再加1, 该位置的值就变成了65, 输出该位置的值就是`Ascii`字符`A`.

```
,.,.,.
abc
```

Output:

```
abc
```

解释: 在本样例中, 首先通过`,`读入一个字符, 然后直接将这个字符输出. 重复三次, 第二行是数据读入, 本样例中依次读入`a`, `b`, `c`并输出.

提示

本题中需要先读取brainfuck程序, 由于程序只有一行, 因此可以使用`getline`直接读取, `,`操作只读取单个字符, 因此可以使用`char input = getcahr()`获取单个字符.

数据范围

输入的Brainfuck代码长度不超过1000.

提交格式

你提交的文件结构应该类似如下形式:

```
<your student number>.zip
|- 1_queuing
|  |- main.cpp
|
|- 2_dynamicQueue
|  |- main.cpp
|
|- 3_reversi
|  |- Chess.h
|  |- Chess.cpp
|  |- main.cpp
|
|- 4_brainfuck
|  |- main.cpp
|
|- cs1604.txt (include it if you use StanfordCppLib)
```

你可以通过以下命令来使用judger

```
python judger.py -T 1_queuing -S your_source_folder \
          -I data/1_queuing/1.in -O data/1_queuing/1.out

python judger_batch.py -T 1_queuing -S your_source_folder
```