二次元配列の問題(Benesse 模試)　　３年（　　）組（　　）番　名前（　　）

第３問 次の問い（問１～３）に答えよ。（配点　25）

テーブルゲーム部のＴさんは，２週間後に迫った今年度の文化祭での出し物として，次のようなゲームで遊んでもらうことにした。このゲームでは，８×８の正方形のマス目が描かれた盤と，表裏を黒と白に塗り分けた平たい石を使用し，ゲーム開始時点での盤面は図１のとおりとする。ゲームは黒と白をそれぞれ担当するプレイヤー２人が交互に盤面へ石を置いていき，相手の石を自分の石で挟んだとき，相手の石を裏返すことで自分の石にする。これを繰り返すことでゲームが進行する。また，ゲーム終了となるルールは，すべてのマスに石が置かれたとき，または，空きマスがあっても両者ともに挟める石がないときとし，最終的に盤上の石が多かった方が勝ちとなる。

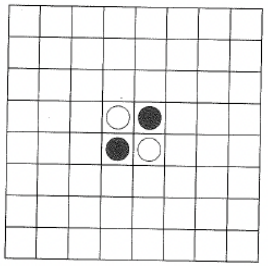

図１　ゲーム開始時点での盤面

次に，Ｔさんはどちらのプレイヤーが勝ったかがすぐわかるようにするため，盤面を映すカメラを設置し，自動的に石の色を判別して盤面を読み取り，二次元配列に取り込めるようにした。

【二次元配列の説明と例】

図２のように，配列名が A で，行，列それぞれを添字 0 ～ 2 で指定する二次元配列の各要素を

配列名［行の添字］［列の添字］

という形式で表す。例えば，図２の中で，二次元配列 A の，添字 2 の行かつ添字 1 の列の要素は A[2][1] となる。

配列 A	0	1	2
0	A[0][0]	A[0][1]	A[0][2]
1	A[1][0]	A[1][1]	A[1][2]
2	A[2][0]	A[2][1]	A[2][2]

図２　二次元配列のイメージ

問１　次の文章の空欄 ア ～ ウ に当てはまる数字をマークせよ。

図３はこのゲームが終了したときの盤面例である。

盤面の読み取りでは，上下方向の位置を行の添字０～７で指定し，左右方向の位置を列の添字０～７で指定することで，盤面のすべてのマスの位置の要素を二次元配列として格納することができる。二次元配列の名前を Kekka とし，行の添字を変数 tate，列の添字を変数 yoko とすると，格納された要素は Kekka[tate][yoko] として表され，例えば，添字１の行（tate = 1）かつ添字 4 の列（yoko = 4）の要素は，Kekka[1][4] で表すことができる。

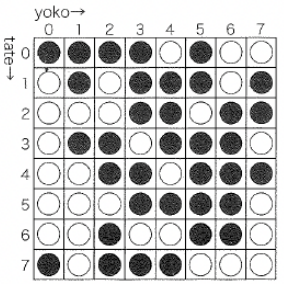

図３　ゲーム終了時の盤面例

二次元配列 Kekka には，初期値としてすべての要素に 0 を格納しておき，二次元配列 Kekka にデータを取り込む際は，「黒の石を 0，白の石を 1」と変換して各要素を更新していく。ただし，石が置かれていない場合，二次元配列 Kekka の要素は更新しない。

なお，読み取ったデータは，次のように二次元配列に格納される。

配列名 = [[添字 0 の行の要素をカンマ区切り], [添字 1 の行の要素をカンマ区切り], …]

例えば，図３のゲーム終了時の盤面例では

Kekka = [[0, 0, 0, 0, 1, 0, 1, 1], [1, 0, 1, 0, 0, 0, 1, 0], …
…, [0, 1, 0, 0, 0, 1, 1, 1]]

と格納される。このとき，二次元配列 Kekka において，Kekka[1][4] = 0 であり，Kekka[4][1] = ア である。また，二次元配列 Kekka において，添字１の行の要素のうち１の個数は イ 個であり，添字２の行の要素のうち０の個数は ウ 個である。

問2　次の文章を読み，空欄 エ ，クケ に当てはまる数字をマークせよ。また，空欄 オ ～ キ に入れるのに最も適当なものを，後の解答群のうちから一つずつ選べ。ただし，空欄 カ ・ キ は同じものを選んでもよい。

盤面のデータを正確に格納できることを確認したTさんは，次に，ゲーム終了時の盤面に応じて，集計を自動化する方法を考えることにした。そこでできあがったプログラムが，すべてのマスが埋まったときにそれぞれの石の個数を集計するためのプログラム1（図4）である。

なお，プログラムの作成にあたり，黒の石の個数を表す変数 kuro，白の石の個数を表す変数 shiro と，カウント用として変数 i が定義されている。

(01)行目では，図3のようなゲーム終了時の盤面に応じたデータが読み取られ，二次元配列 Kekka に格納される。

(02)～(04)行目では，定義した変数を初期化する処理がされる。

(05)～(08)行目では，二次元配列から行ごとに値を順に取り出し，白の石の個数をカウントすることを繰り返している。

(09)行目では，得られたデータから黒の石の個数を算出する。

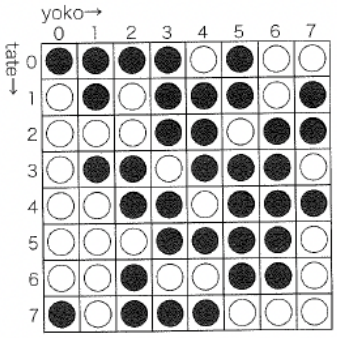

図3　ゲーム終了時の盤面例（再掲）

```
(01) Kekka = [[ 0, 0, 0, 0, 1, 0, 1, 1 ], (以下, 省略) … ]
(02) shiro = 0
(03) kuro = 0
(04) i = 0
(05) tate を 0 から [エ] まで 1 ずつ増やしながら繰り返す：
(06)     yoko を 0 から [エ] まで 1 ずつ増やしながら繰り返す：
(07)         shiro = [オ]
(08)         i = i + 1
(09) kuro = [カ] - [キ]
```

図4　プログラム1

いま，Tさんが図3の盤面例に対してプログラム1を実行したとき，最終的に i に格納されている値は クケ である。

オ の解答群

⓪ shiro + 1
① shiro + i
② shiro + Kekka [tate][yoko]
③ shiro + Kekka [tate + 1][yoko + 1]

カ ・ キ の解答群

⓪ kuro　　① shiro
② 1　　③ i
④ Kekka [tate][yoko]　　⑤ Kekka [tate + 1][yoko + 1]

問3　次の文章を読み，空欄［コ］～［シ］に入れるのに最も適当なものを，後の解答群のうちから一つずつ選べ。

次にTさんは，すべてのマスが埋まったときに，石の個数を基にどちらのプレイヤーが勝利したかを判別するため，プログラム1を改変してプログラム2（図5）を完成させた。

(10)～(15)行目では，色に対応する石の個数を格納した変数を用い，その大小関係を比較することで表示する勝利プレイヤーを場合分けするものである。また，(16)行目では，次のゲーム結果の読み取りに備え，二次元配列 Kekka の要素をすべて 0 に戻すという初期化処理を行うものである。

```
(01) Kekka = [[ 0, 0, 0, 0, 1, 0, 1, 1 ], (以下, 省略) … ]
(02) shiro = 0
(03) kuro = 0
(04) i = 0
(05) tate を 0 から［エ］まで 1 ずつ増やしながら繰り返す：
(06)     yoko を 0 から［エ］まで 1 ずつ増やしながら繰り返す：
(07)         shiro = ［オ］
(08)         i = i + 1
(09) kuro = ［カ］ - ［キ］
(10) もし［コ］ならば：
(11)     「黒の勝ち」と表示する
(12) そうでなくてもし［サ］ならば：
(13)     「引き分け」と表示する
(14) そうでなければ：
(15)     「白の勝ち」と表示する
(16) 二次元配列 Kekka のすべての要素を 0 として初期化する
```

図5　プログラム2

Tさんができあがったプログラム2を図3の盤面に対して実行したところ，「黒の勝ち」と正しく表示された。こうして，ゲーム終了時にすべてのマスが埋まった盤面の読み取りと石の個数の集計，勝利プレイヤーの表示を自動処理するプログラムができた。

しかし，文化祭まであと1週間となった活動日，起こり得るゲーム終了時の盤面例で最終テストをしていたところ，すべてのマスが埋まらない場合に，勝利プレイヤーが正確に判別できない「［シ］」というケースが発見された。そこで，文化祭本番に間に合わせるため，Tさんはプログラムの改変に取りかかることにした。

［コ］・［サ］の解答群

⓪ kuro > shiro	① kuro < shiro	② kuro == shiro
③ i > kuro	④ i > shiro	⑤ i < kuro
⑥ i < shiro	⑦ i == kuro	⑧ i == shiro

［シ］の解答群

⓪　黒の石の個数と白の石の個数が同数のときに，「白の勝ち」と表示される

①　黒の石の個数が白の石の個数より多いときに，「白の勝ち」と表示される

②　黒の石の個数が白の石の個数より少ないときに，「黒の勝ち」と表示される

③　黒の石の個数が白の石の個数より多いときに，「引き分け」と表示される