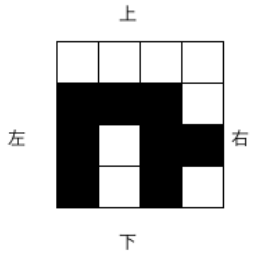
1　課題説明

今回の本選課題はテトリスという有名なゲームにヒントを得たものである。オリジナルのテトリスと違い、使うピースのセットがあらかじめ与えられる。それらを (1) どのような順番で (2) どの向きで (3) どこに落とせば、最終的に積み上がった高さを最も低くできるかを競う。

1.1　ピース

ピースは正方形がいくつかつながってできた平面図形である。各正方形は同一ピース内の他の正方形の少なくともひとつと一辺以上を共有する。たとえば



の黒い正方形の集まりが，1 つのピースである．どのピースも正方形 4 × 4 マスの中に収まる．4 × 4 マスをピースの枠と呼ぶ．最小のピースは正方形ひとマス，最大は正方形 16 マスである．なお，内部に穴の開いたピースなどもあることに注意．ピース枠には図のように上下左右がある

1.2　ゲームの盤面

盤面は、横幅がピースを構成する正方形の整数倍に決まっており、縦の長さには制限がない。盤面の上下左右は図のように決める。

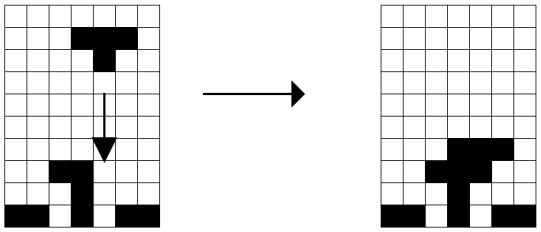
上
左　右
下

以下では、横の一行を「段」、縦の一行を「列」と呼ぶ。盤面の最左列を「第 0 列」とし、以下、右に向かって「第 1 列」「第 2 列」と番号をつける。また，盤面の最下段を「第 0 段」，それから上に向かって「第 1 段」,「第 2 段」と番号をつける

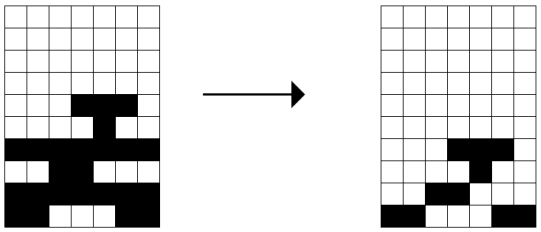
1.3　ゲームのルール

ルールは基本的にテトリスと同じであるが、いくつか簡略化した点がある。

1. ピースはひとつずつ盤面に出現する。出現位置は盤上に存在するどの正方形よりも上方である (無限上方と思ってよい)。
2. ピースは出現時に，回転させて，(a) 上下左右のどの辺を下にするか，(b) どの列（の左端）にピース枠の左端を合わせるか（出現位置列番号）を決める．その出現列と向きを保ったまま，ピースは，すでに落ちて積み上がっている正方形と重ならない限りいちばん低い位置まで落ちる．



3. 盤面のどの高さでも、横一段すべてが正方形で埋まると、その段の正方形はすべて消え、その段より上にある正方形はすべて一段分だけ下に平行移動する。



4. 問題に与えられたすべてのピースを使い終えたとき、最終的に最も高い位置にある正方形の高さ（その正方形が入る段の番号）が得点である。

(注) 高さに制限はなく、ピースがあるかぎり、どこまで積み上がってもよい。また、ピースの出現時に枠のどの辺を下にするかと枠の左辺の位置を決めたら、あとはそのままま っすぐ下に落ちるだけである。オリジナルのテトリスのように、落ちている最中に横に動かすといった操作はできない

課題問題

問題として、あらかじめ決められた数のピースのセットが配られる (たとえば、100 個のピースが配られる。その中にはもちろん、同じ形のピースが複数含まれる)。それらのピースをすべて使ってゲームをするとき、得点がなるべく低くなるように、(1) ピースを落とす順番 (2) 各ピースの向きと出現位置列番号を決めて欲しい.

2 課題の詳細

2.1 問題のサイズ

ピースの種類: 15 種類
盤面の横幅 (一段の幅)： 15 列（正方形 15 マス分）
ピースの数： 100 個から 200 個
制限時間: 問題の読み込み終了時からの経過時間 10 分までに出力された最後の結果を解答とする

2.2 問題の入力

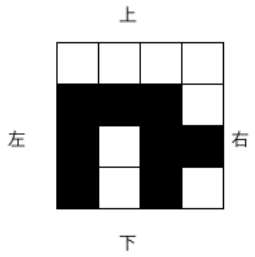
ピースは別図に示した 15 種類を用いる。それぞれのピースには ID 番号が割り当てられている。ピースのセットはファイル、`sc_pieces.dat` で与えられる。ファイルの各行には

`ID` **ピースの形**

の順で数字が書かれている。ピースの形は 0 と 1 の 16 個の並びで表現される。1 がピースの構成要素である■，0 は空白を表す□に対応する．たとえば一行目が

```
0 1010101111100000
```

なら、ID 番号 0 のピースが



という意味である。

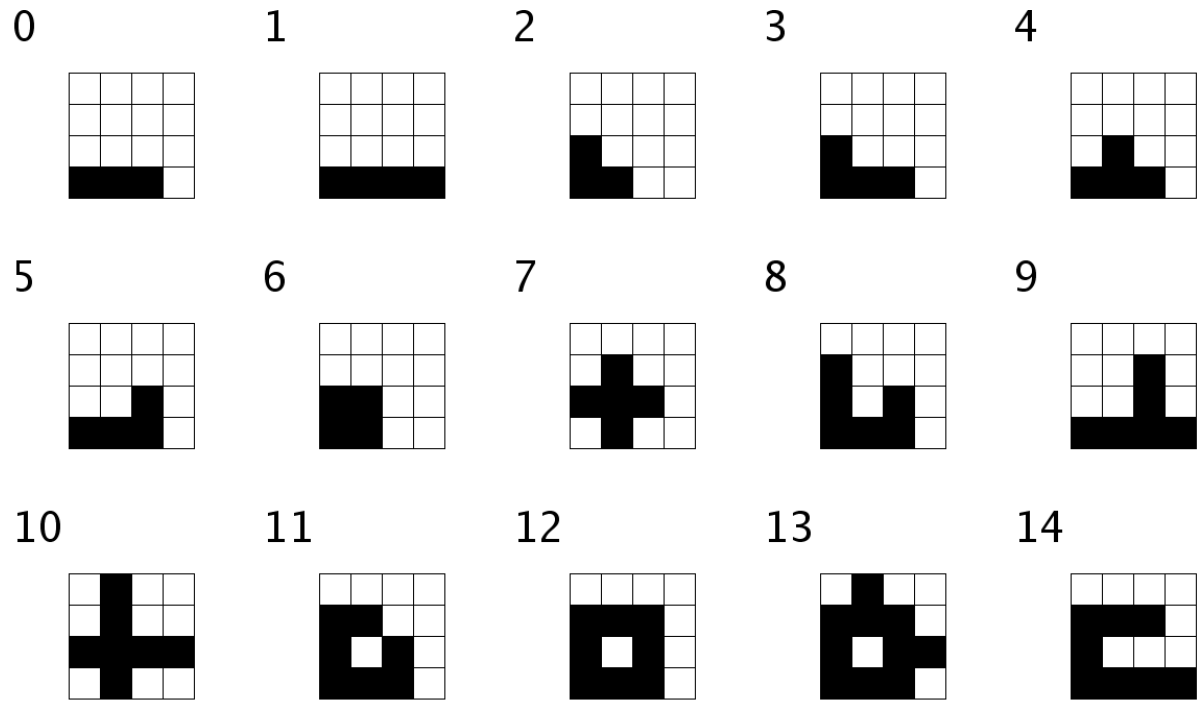


図 1　使用するピース一覧 (番号が ID)

2.3　問題の与え方

問題は各ピースの個数のセットとして与えられる。たとえば、ID=0 のピースを 8 個、1 のピースを 5 個、2 のピースを 5 個・・・、14 を 3 個の場合、問題の入力ファイルには

```
0 8
1 5
2 5
......
14 3
```

と ID の昇順に書かれる。ピースのセットと問題は、用意された関数 `sc_input()` をプログラムの最初に呼ぶことで読み込まれる。これについては後で詳しく説明する。

2.4 解答の出力

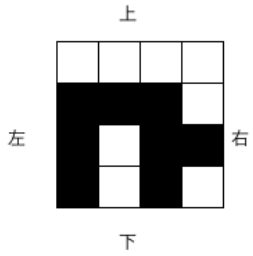
解答は、ピースを使用する順に

ID　**回転　位置**

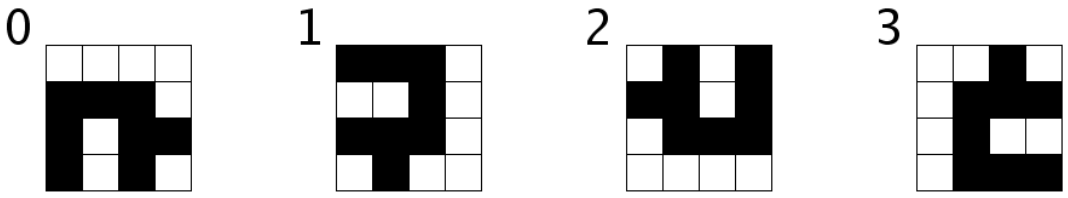
のセットを出力する。使用するピースの数が出題と一致していない場合は失格である。なお、後で詳しく説明するように、出力には用意された関数 sc_output() を使う。解答は何度出力してもよく、制限時間以内で最後に出力されたものを解答として採用する。

2.5 回転について

回転はピース枠の上下左右のどこを下にするかを，0 から 3 までの数字で指定する．下 = 0, 右 = 1，上 = 2，左 = 3 とする．（時計回り順）．0,1,2,3 以外の数字を指定すると 0 とみなされる。たとえば、ピースが



であれば、



となる。回転の方向を間違えないように注意すること。

2.6 位置の指定

出現時にピースを置く位置を，枠の左端を盤面の第何列めに置くかで指定する．枠の左端の列が空白であってもその位置で指定することに注意。従って、位置は-3 から 14 までの数字で指定される。ピースがひとつでも盤面からはみだした場合はそのゲームは失格とする。なお、回転と位置は、先に回転させてから指定の位置に置くという順番であることに注意。

3 ヘッダーファイルと入出力関数

3.1 ヘッダーファイル

プログラムには必ず

```
#include "sc14.h"
```

として、ヘッダーファイル sc14.h をインクルードする。sc14.h には入出力用の関数およびいくつかの定数、変数、配列が定義されている。なお、これらの変数と配列はグローバルに定義されている。ヘッダーファイルを解析するのは自由だが、決して改変してはならない (採点時には採点用の別のヘッダーファイルが使われる)。

3.2 定義されている定数

SC_W: 盤面の幅で 15 である
SC_NP: ピースの種類で 15 である
SC_NMAX: ピースの数の最大値で 200 である
SC_INPUT_FILE: ピースのセットを格納したファイル名で、sc_pieces.dat である

3.3 定義されている変数

int sc_num: sc_input() を呼ぶと、問題で実際に使うピースの数が格納される
int sc_starttime: sc_input() を呼ぶと、問題入力が終了した時刻が格納される
int sc_outputtime: sc_output() を呼ぶと、その時点での時刻が格納される。

これらの変数は sc_input() あるいは sc_output() を呼んだ時点で値が代入される。プログラム中ではこれらの変数を参照するだけとし、勝手に数値を代入しないこと

3.4 定義されている配列

int sc_piece[SC_NP][16]: sc_input() を呼ぶと、ピースの ID を引数として、ピースの形を 0 と 1 で表した文字列が格納される。id 番目のピースは sc_piece[id][0] から sc_piece[id][15] までの 16 個の配列に格納されている。ファイル sc_pieces.dat から読み込まれる。

int sc_prob[SC_NP]: sc_input() を呼ぶと、問題が格納される。ID=n のピースの個数が sc_prob[n] に格納される。標準入力から読み込まれる。

int sc_result[SC_NMAX][3]: 解答を格納する配列である。i 番目に使うピースの ID、回転、位置をそれぞれ sc_result[i][0]、sc_result[i][1]、sc_result[i][2] に格納したのちに出力関数 sc_output() を呼ぶ。i は 0 から sc_num までである。

3.5 入力関数

プログラムの最初の実行文は

```
sc_input();
```

とすること (この前には定義文以外を書かないこと)。これにより、ピースの定義と問題が読み込まれ、開始時刻が記録される。ピースの定義は sc_pieces.dat から、また、問題は標準入力から読み込まれる。なお、 sc_input() は必ずランク 0 で 1 回のみ実行すること (ランクについては MPI の解説を参照のこと)

3.6 出力関数

解答を出力する際には、配列 sc_result に解答を格納してから、関数

```
sc_output();
```

を呼ぶと、解答が

```
START
経過時間
 sc_result[0][0] sc_result[0][1] sc_result[0][2]
 sc_result[1][0] sc_result[1][1] sc_result[1][2]
 sc_result[2][0] sc_result[2][1] sc_result[2][2]
......
 sc_result[sc_num-1][0] sc_result[sc_num-1][1] sc_result[sc_num-1][2]
END
```

のように出力される。解答は標準出力に出力される。なお、sc_output() は必ずランク 0 でのみ実行すること。出力は何度行ってもよいが、制限時間内の最後に出力されたものが解答として採用される。

4 提出する解答プログラム

解答プログラムは複数に分割せず、必ずひとつのプログラムであること。独自のヘッダーファイルの使用なども認めない。また、makefile の使用は認めない。sc14.h ヘッダーファイルは不要である。採点時には gcc の -O2 -m オプションのみをつけてコンパイルする。提出のしかたは、当日の説明にしたがうこと

5 乱数

プログラム中で乱数が必要になる場合もあるだろう。そのときは GCC の stdlib ライブラリーにある rand() 関数を使う。そのためにヘッダー stdlib.h が必要である。rand() 関数は、呼ばれるたびに 0 から RAND_MAX (stdlib 内で定義されている定数) までの整数をランダムに返す。乱数は「初期化」しないと常に同じ順番で数が出てしまう。初期化のためには srand(unsined int seed) 関数を使う。seed として整数を与えると、seed の値ごとに異なる乱数の系列が出力される。srand はプログラム中で乱数を使用する前に一度だけ呼べばよい。たとえば

```
#include<stdio.h>
#include<stdlib.h>
main(){
int i,r;
unsigned int seed;
seed = 12345;
srand(seed);
for(i=0; i<10; ++i){r = rand(); printf("%d\n",r);}
}
```

を実行すれば 10 個の乱数が出力され、seed の 12345 を別の数にすれば別の乱数が出力される。

なお、並列計算の場合、各スレッドごとに異なる seed を与えないと、すべてのスレッドで同じ乱数列が出てしまうので注意が必要である。乱数を使う場合には、まずは乱数を生成するだけの簡単なプログラムを書いて、試してみることを薦める

6 経過時間の取得

プログラム途中で経過時間を知りたいことがあるかもしれない。開始時刻は変数 sc_starttime に格納されている。任意のタイミングで経過時間を知りたいときは

```
(int)time(NULL) - sc_starttime
```

で経過時間が秒単位で得られる。なお、時刻の関数を使うために必要なヘッダーファイル time.h は sc14.h の中ですでにインクルードされているので、改めてインクルードしなくてよい。