Задача 1. Командная олимпиада

Входной файл	input.txt
Выходной файл	output.txt
Ограничение времени (сек/тест)	1
Ограничение памяти (Мб)	64
Максимальный балл	25+10 (*)

Условие задачи

Согласно правилам городской командной олимпиады по программированию каждая команда должна состоять ровно из трех человек. Для решения множества задач каждой команде предоставляется только один компьютер, поэтому для успешного участия в такой олимпиаде работа членов команды во время олимпиады должна быть согласована и хорошо отлажена.

В гимназии есть три физико-математических класса, в которых обучаются только талантливые ученики с хорошими знаниями информатики и навыками программирования. Учителю информатики надо сформировать команды из учеников этих классов для участия в олимпиаде. По мнению учителя, в каждой команде должны быть либо ученики одного и того же класса, либо по одному ученику из всех трех классов одновременно. (Если в составе команды будет два ученика из одного класса и один из другого, то велика вероятность, что первые два, объединившись, начнут «притеснять» третьего, что приведет к конфликтам и спорам внутри команды во время проведения олимпиады, что недопустимо).

Необходимо найти максимальное количество команд, которые может сформировать учитель информатики для участия в олимпиаде с учетом указанных выше принципов формирования команд.

Входные данные

Во входном файле в первой строке через пробел записаны три целых числа X, Y и Z – кол-во учеников в трех физико-математических классах гимназии ($0 \le X, Y, Z \le 10^9$).

Выходные данные

В выходной файл необходимо записать единственное целое число N – максимальное количество команд, которые могут быть сформированы в гимназии согласно указанным выше правилам.

Пример входного файла (input.txt)	**Пример выходного файла (output.txt)**
18 21 24	21
22 26 22	23
0 0 0	0

(*) – дополнительные 10 баллов начисляются за абсолютно правильное решение задачи (решение считается абсолютно правильным, если оно проходит все тесты)

Решение жюри

```
#include <fstream>
#include <algorithm>

using namespace std;


int main()
{
  ifstream cin("input.txt");
  ofstream cout("output.txt");

  int x, y, z;
  cin >> x >> y >> z;

  int min_colors_count = min(x, min(y, z));
  int n = min_colors_count;
  x -= n;
  y -= n;
  z -= n;

  n += x / 3;
  x %= 3;
  n += y / 3;
  y %= 3;
  n += z / 3;
  z %= 3;

  if (x + y + z == 4 && min_colors_count)
    n++;

  cout << n << endl;
}
```

Задача 2. Пепси-Кола

Входной файл	input.txt
Выходной файл	output.txt
Ограничение времени (сек/тест)	1
Ограничение памяти (Мб)	64
Максимальный балл	25+10

Условие задачи

Бутылка Пепси-Колы стоит ***K*** рублей, а пустая бутылка – ***L*** рублей. В понедельник несколько друзей купили несколько бутылок Пепси-Колы на все имевшиеся у них ***N*** рублей. На следующий день Пепси-Колы у них уже не осталось. Они сдали пустые бутылки, добавили сдачу предыдущего дня и снова на все деньги купили Пепси-Колы. Данная процедура продолжалась каждый день, пока у друзей хватало денег.

Требуется найти минимальную сумму N, при которой в пятницу друзьям можно будет купить себе хотя бы одну бутылку Пепси-Колы.

Входные данные

Входной файл содержит два целых числа K и L ($1 \le K, L \le 2 * 10^9$), записанные в одной строке через пробел. Гарантируется, что минимальная сумма, которая должна быть у друзей в понедельник, также меньше $2 * 10^9$.

Выходные данные

В выходном файле необходимо записать одно целое число N – минимальную сумму денег, имевшуюся у друзей в понедельник, такую, чтобы друзья могли купить Пепси-Колы в пятницу.

Пример входного файла (input.txt)	**Пример выходного файла (output.txt)**
5 1	629

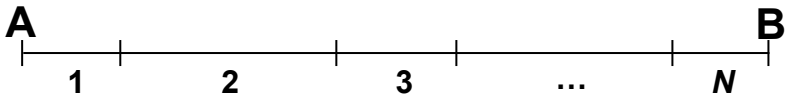
Решение жюри

```
#include <fstream>
#include <algorithm>

using namespace std;



int main()
{
  ifstream cin("input.txt");
  ofstream cout("output.txt");

  int x, y, z;
  cin >> x >> y >> z;

  int min_colors_count = min(x, min(y, z));
  int n = min_colors_count;
  x -= n;
  y -= n;
  z -= n;

  n += x / 3;
  x %= 3;
  n += y / 3;
  y %= 3;
  n += z / 3;
  z %= 3;

  if (x + y + z == 4 && min_colors_count)
    n++;

  cout << n << endl;
}
```

Задача 3. Время встречи

Входной файл	input.txt
Выходной файл	output.txt
Ограничение времени (сек/тест)	1
Ограничение памяти (Мб)	64
Максимальный балл	25+10

Условие задачи



Между двумя городами A и B имеется дорога, состоящая из N ($1 \le N \le 1000$) участков. Участки нумеруются, начиная от города A до города B. Мотоцикл i-ый участок проезжает за время a_i, машина – за время b_i ($1 \le a_i, b_i \le 1000$; a_i, b_i – целые). Одновременно мотоцикл и машина начинают движение: мотоцикл из города A в город B, машина из города B в город A. Требуется найти время, через которое мотоцикл и машина встретятся.

При решении задачи следует считать, что на каждом отдельном участке и мотоцикл и машина движутся равномерно.

Входные данные

В первой строке входного файла записано число N. Во второй строке через пробел записаны числа $a_1, a_2, \ldots, a_N$, в третьей – числа $b_1, b_2, \ldots, b_N$.

Выходные данные

В выходной файл необходимо записать одно вещественное число – время, через которое произойдет встреча, с точностью не менее 10^{-6}. В качестве символа-разделителя целой и дробной части использовать точку.

Пример входного файла (input.txt)	**Пример выходного файла (output.txt)**
3 3 1 4 7 1 3	3.500000
3 2 3 1 2 3 1	3.000000
1 7 4	2.545455

Решение жюри

```
#include <fstream>
#include <vector>
#include <ios>
#include <iomanip>
#include <algorithm>

using namespace std;


int main()
{
  ifstream cin("input.txt");
  ofstream cout("output.txt");

  int n;
  cin >> n;

  vector<int> a(n), b(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];
  for (int i = 0; i < n; i++)
    cin >> b[i];

  double result = 0;
  int a_i = 0, b_i = n;
  double a_t = 0, b_t = 0;
  while (true) {
    if (b_i == a_i) {
      result = a_t;
      break;
    }
    else if (b_i - a_i == 1) {
      result =
        a_t > b_t ? a_t + (b[a_i] + b_t - a_t) * a[a_i] / (a[a_i] + b[a_i])
                  : b_t + (a[a_i] + a_t - b_t) * b[a_i] / (a[a_i] + b[a_i]);
      break;
    }
    else {
      int temp = (a_t + a[a_i]) - (b_t + b[b_i - 1]);
      if (temp >= 0) {
        b_t += b[b_i - 1];
        b_i--;
      }
      if (temp <= 0) {
        a_t += a[a_i];
        a_i++;
      }
    }
  }

  cout << fixed << setprecision(6) << result << endl;
}
```

Задача 4. Построение по росту

Входной файл	input.txt
Выходной файл	output.txt
Ограничение времени (сек/тест)	1
Ограничение памяти (Мб)	64
Максимальный балл	25+10

Условие задачи

На первом уроке физкультуры в математическом классе при построении в шеренгу ученики встали в случайном порядке. Учителю такой порядок учеников не понравился и он решил построить учеников по росту (первым в шеренге должен стоять самый высокий ученик). Так как класс был математический (учитель физкультуры тоже когда-то учился в математическом классе), то учитель решил усложнить задачу. Построение по росту должно осуществляться с помощью обменов. На каждом шаге перестроения два ученика, один из которых обязательно юноша, выходят из шеренги и меняются местами. С учетом наложенных выше условий построение по росту, согласно требованиям учителя, должно осуществиться не более чем за $2 * N$ обменов, где $\boldsymbol{N}$ – количество учеников в классе (при этом оптимальности обменов, т.е. их наименьшего количества, учитель не требует, важно, чтобы количество обменов было не более заданного).

Гарантируется, что хотя бы один юноша на уроке физкультуры присутствует.

Если рост нескольких учеников одинаковый, то в построенной по росту шеренге между собой они могут стоять в произвольном порядке (пол участников значения не имеет).

Необходимо найти любую, удовлетворяющую требованиям учителя физкультуры, последовательность обменов учеников для построения шеренги по росту.

Входные данные

В первой строке входного файла записано число N $(1 \le N \le 1000)$. В последующих N строках содержится информация об учениках класса в том порядке, в котором они первоначально построились. В каждой из этих строк через пробел записаны пол ученика в виде одного символа ('B' – юноша, 'G' – девушка) и его рост в виде целого числа $\boldsymbol{H_i}$ $(100 \le H_i \le 200)$.

Выходные данные

В первой строке выходного файла необходимо записать число $\boldsymbol{M}$ – кол-во необходимых обменов учеников для построения шеренги по росту. Последующие M строк описывают M обменов. В каждой такой строке необходимо записать через пробел два целых числа – позиции двух учеников, которые на данном шаге перестроения меняются местами. Позиция ученика – порядковый номер ученика в шеренге в тот момент времени, когда эта позиция определяется. Нумерация начинается с единицы (позиция первого ученика в шеренге – 1, последнего – N).

Пример входного файла (input.txt)	**Пример выходного файла (output.txt)**
6 G 155 B 170 B 155 G 175 B 168 B 163	4 1 5 1 4 3 6 4 3

Решение жюри

```
#include <fstream>
#include <vector>
#include <algorithm>

using namespace std;


typedef struct {
  bool boy;
  int height;
  int pos, from, to;
} student_t;


bool height_comparator(const student_t &a, const student_t &b) {
  return a.height > b.height;
}

bool is_boy(const student_t &a) {
  return a.boy;
}


bool is_sorted(const vector<student_t> &students) {
  for (unsigned int i = 1; i < students.size(); i++)
    if (students[i - 1].height < students[i].height)
      return false;
  return true;
}


void swap_students(vector<student_t> &students, int i, int j) {
  swap(students[i], students[j]);
  swap(students[i].from, students[j].from);
  swap(students[students[i].to].from, students[students[j].to].from);
}


typedef struct {
  unsigned int i, j;
} change_t;



int main()
{
  ifstream cin("input.txt");
  ofstream cout("output.txt");

  int n;
  vector<student_t> students;

  cin >> n;
  for (int i = 0; i < n; i++) {
    student_t student;
    char ch;
    cin >> ch >> student.height;
    student.boy = ch == 'B';
    student.pos = i;
    students.push_back(student);
  }

  if (is_sorted(students))
    cout << "0" << endl;
  else {
    vector<student_t> sorted = students;
```

```
    sort(sorted.begin(), sorted.end(), height_comparator);
    for (int i = 0; i < n; i++) {
      students[i].from = sorted[i].pos;
      students[sorted[i].pos].to = i;
    }
    sorted.clear();

    int boy_pos = find_if(students.begin(), students.end(), is_boy)->pos;


    vector<change_t> changes;

    for (int i = 0; i < n; i++)
      if (i != students[boy_pos].to && i != students[i].from) {
        int j = students[i].from;
        if (students[i].boy || students[j].boy) {
          swap_students(students, i, j);
          change_t change = { min(i, j), max(i, j) };
          changes.push_back(change);
          if (boy_pos == i)
            boy_pos = j;
          else if (boy_pos == j)
            boy_pos = i;
        }
        else {
          swap_students(students, i, boy_pos);
          change_t change1 = { min(i, boy_pos), max(i, boy_pos) };
          changes.push_back(change1);
          swap_students(students, i, j);
          change_t change2 = { min(i, j), max(i, j) };
          changes.push_back(change2);
          boy_pos = j;
        }
      }

    cout << changes.size() << endl;
    for (unsigned int i = 0; i < changes.size(); i++)
      cout << changes[i].i + 1 << " " << changes[i].j + 1 << endl;
  }
}
```

Задача 5. Игра в шашки

Входной файл	input.txt
Выходной файл	output.txt
Ограничение времени (сек/тест)	1
Ограничение памяти (Мб)	64
Максимальный балл	25+10

Условие задачи

В прошлом году ученик физико-математического класса Вовочка разработал компьютерную программу игры в шахматы. Однако сложность задачи оказались слишком большой, поэтому получившаяся программа «играла» очень «слабо». С учетом полученного опыта Вовочка решил, что игра в шашки гораздо легче игры в шахматы, поэтому реализовать «сильный» алгоритм игры в шашки ему вполне по силам.

Одной из множества частных задач, выделенных Вовочкой из основной задачи, является подсчет количества возможных ходов, которые можно сделать каждой шашкой в текущий момент игры. Вовочка просит вас помочь ему решить данную задачу.

Существует несколько вариантов правил игры в шашки, Вовочка выбрал вариант правил «Русские шашки». Эти правила широко известны и поэтому приводить их в условии данной задачи не имеет смысла (если кто-то этих правил не знает, можно воспользоваться информацией из википедии https://ru.wikipedia.org/wiki/Русские_шашки, а также https://www.gambler.ru/Правила_русских_шашек или из других источников).

Однако стоит обратить внимание участников олимпиады на некоторые пункты правил:

- взятие (бой) шашек противника, если есть такая возможность, обязательно;
- при этом вариант взятия определяется игроком;
- взятые (побитые) шашки снимаются с доски по завершении очередного хода;
- при взятии применяется правило турецкого удара – за один ход шашку противника можно побить только один раз, т.е., если при бое нескольких шашек противника шашка или дамка повторно выходит на уже побитую шашку, то ход останавливается;
- при бое через «дамочное» поле (клетка на самой дальней относительно начальной расстановки шашек для игрока линии клеток) простая шашка превращается в дамку и продолжает бой по правилам дамки.

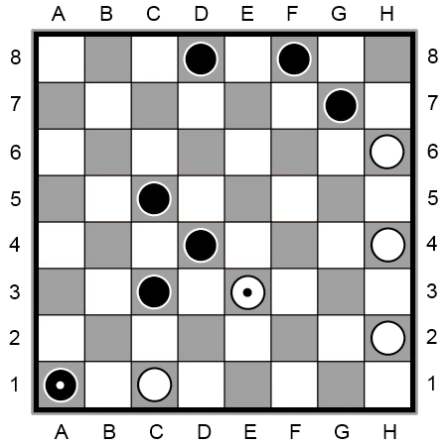
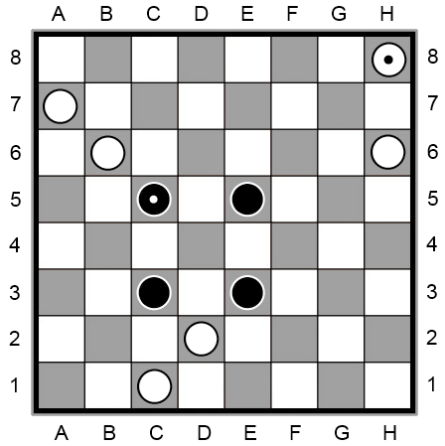
Входные данные

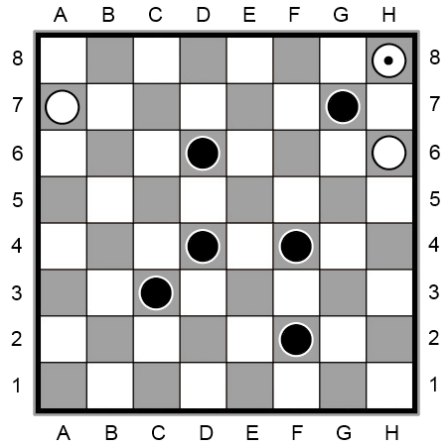
Во входном файле записана информация о расстановке шашек на доске. Каждая клетка доски описана одним символом, т.е. в файле записано 8 строк по 8 символов. Пустым клеткам соответствует символ «точка» ('.'), белые шашки кодируются символами 'w' (обычная шашка) и 'W' (дамка), черные шашки – символами 'b' и 'B' (дамка). Расстановке шашек в начале игры соответствует входной файл, где черные шашки присутствуют в трех первых строках входного файла, а белые шашки – в трех последних строках файла (если смотреть на изображение доски в примерах ниже, линия доски с номером 8 на картинке будет описываться первой строкой входного файла).

Выходные данные

В выходной файл необходимо для каждой белой шашки (включая дамки) записать количество различных ходов, которые можно осуществить данной шашкой в соответствии с правилами игры в Русские шашки при расстановке шашек на доске, указанной во входном файле. Два хода считаются разными, если в результате этих ходов получаются разные расположения шашек на доске.

Порядок вывода количества ходов для белых шашек соответствует порядку описания этих шашек во входном файле, если данный файл читать посимвольно. Т.е. нумерация шашек осуществляется сверху вниз, а в строке слева направо согласно описанию во входном файле (или на соответствующей иллюстрации). Для первого примера входных данных порядок белых шашек следующий: шашка на линии с номером 6 на изображении, затем шашка на линии с номером 4, затем дамка (помечена точкой), затем шашки на линиях с номерами 2 и 1.

Пример входного файла (input.txt)	**Пример выходного файла (output.txt)**	**Комментарий**
...b.b.. .....b. .....w ..b..... ...b...w ..b.W... .....w B.w.....	0 1 5 1 2	
.....W w..... .w.....w ..B.b... ....b.b... ...w.... ..w.....	4 0 3 0 1 0	 Шашка в позиции D2 может взять все черные шашки, обойдя их как по часовой стрелке, так и против. Однако оба этих варианта будут считаться за один ход, т.к. в результате получится одно и то же расположение шашек на доске.

<table>
<tr>
<td>.....W
w.....b.
...b...w
.....
...b.b..
..b.....
.....b..
.....</td>
<td>4
0
6</td>
<td>


- H8:E5:C7
- H8:E5:B8
- H8:E5:G3:E1:B4:E7
- H8:E5:G3:E1:B4:F8

Шашка A7 сделать ход не может, т.к. другие белые шашки могут бить шашки соперника.

Шашка H6, взяв черную шашку G7, превратится в дамку и продолжит взятие шашек соперника как дамка, т.е. для H6 возможны ходы:

- H6:F8:C5:E3:G1
- H6:F8:C5:E3:G5
- H6:F8:C5:E3:H6
- H6:F8:B4:D2:G5
- H6:F8:B4:D2:H6
- H6:F8:B4:E1:G3:E5

В последнем варианте хода после попадания на клетку E5 побить последнюю шашку соперника в клетке D4 нельзя, т.к. этому препятствует уже побитая шашка в клетке C3 (которая снимается только по завершении хода).
</td>
</tr>
</table>

Решение жюри

```
#include <fstream>
#include <sstream>
#include <array>
#include <set>
#include <vector>
#include <iomanip>

using namespace std;


const int CB_SIZE = 8;


typedef array<array<char, CB_SIZE>, CB_SIZE> checkers_board_t;

bool operator <(const checkers_board_t &a, const checkers_board_t &b)
{
  for (int r = 0; r < CB_SIZE; r++)
    for (int c = 0; c < CB_SIZE; c++) {
      if (a[r][c] == b[r][c])
        continue;
      return a[r][c] - b[r][c] > 0;
    }
  return false;
}

typedef array<array<int, CB_SIZE>, CB_SIZE> int_board_t;
typedef array<array<vector<string>, CB_SIZE>, CB_SIZE> steps_board_t;

typedef struct {
  int dr, dc;
} direction_t;


direction_t directions[4] = { { -1, -1 }, { -1, 1 }, { 1, -1 }, { 1, 1 } };



template <typename T> inline bool between(T v, T a, T b) {
  return a <= v && v <= b;
}


inline bool is_pos_correct(int row, int col) {
  return between(row, 0, CB_SIZE - 1) && between(col, 0, CB_SIZE - 1);
}


bool next_beat(checkers_board_t &cb, int row, int col,
               set<checkers_board_t> &boards) {
  char checker = cb[row][col];
  bool result = false;

  if (checker != 'w' && checker != 'W')
    return result;

  for (int i = 0; i < 4; i++) {
    for (int j = 1; j <= (checker == 'W' ? CB_SIZE - 1 : 1); j++) {
      int r = row + directions[i].dr * j,
          c = col + directions[i].dc * j;
      int r_next = r + directions[i].dr,
          c_next = c + directions[i].dc;

      if (!is_pos_correct(r_next, c_next))
        break;
```

```
      if (cb[r][c] == 'b' || cb[r][c] == 'B')
        if (cb[r_next][c_next] == '.') {
          result = true;
          char checker_next = r_next == 0 ? 'W' : checker;

          char cb_r_c = cb[r][c];
          cb[r][c] = '_';
          cb[row][col] = '.';

          bool can_beat = false;
          for (int jj = j + 1;
               jj <= (checker == 'W' ? CB_SIZE - 1 : 2); jj++) {
            int rr = row + directions[i].dr * jj,
                cc = col + directions[i].dc * jj;

            if (!is_pos_correct(rr, cc) || cb[rr][cc] != '.')
              break;

            cb[rr][cc] = checker_next;
            can_beat = next_beat(cb, rr, cc, boards) || can_beat;
            cb[rr][cc] = '.';
          }
          if (!can_beat)
            for (int jj = j + 1;
                 jj <= (checker == 'W' ? CB_SIZE - 1 : 2); jj++) {
              int rr = row + directions[i].dr * jj,
                  cc = col + directions[i].dc * jj;

              if (!is_pos_correct(rr, cc) || cb[rr][cc] != '.')
                break;

              cb[rr][cc] = checker_next;
              boards.insert(cb);
              cb[rr][cc] = '.';
            }

          cb[row][col] = checker;
          cb[r][c] = cb_r_c;
        }

        if (cb[r][c] != '.')
          break;
    }
  }

  return result;
}



int main()
{
  ifstream cin("input.txt");
  ofstream cout("output.txt");

  checkers_board_t cb;

  for (int r = 0; r < CB_SIZE; r++)
    for (int c = 0; c < CB_SIZE; c++)
      cin >> cb[r][c];

  bool can_beat = false;
  int_board_t steps_counts;
  steps_counts[0].fill(0);
  steps_counts.fill(steps_counts[0]);

  for (int r = 0; r < CB_SIZE; r++)
    for (int c = 0; c < CB_SIZE; c++) {
      set<checkers_board_t> boards;
```

```
      if (next_beat(cb, r, c, boards)) {
        can_beat = true;
        steps_counts[r][c] = boards.size();
      }
    }

  if (!can_beat)
    for (int r = 0; r < CB_SIZE; r++)
      for (int c = 0; c < CB_SIZE; c++)
        if (cb[r][c] == 'w' || cb[r][c] == 'W')
          for (int i = 0; i < (cb[r][c] == 'W' ? 4 : 2); i++)
            for (int j = 1; j <= (cb[r][c] == 'W' ? CB_SIZE - 1 : 1); j++) {
              int rr = r + directions[i].dr * j,
                  cc = c + directions[i].dc * j;

              if (!is_pos_correct(rr, cc) || cb[rr][cc] != '.')
                break;

              steps_counts[r][c]++;
            }

  for (int r = 0; r < CB_SIZE; r++)
    for (int c = 0; c < CB_SIZE; c++)
      if (cb[r][c] == 'w' || cb[r][c] == 'W')
        cout << steps_counts[r][c] << endl;
}
```