```
 1: /*
 2:  * queens.cc
 3:  *
 4:  *  Created on: Feb 14, 2017
 5:  *      Author: lubo
 6:  */
 7:
 8: #include <vector>
 9: #include <cstdlib>
10: #include <iostream>
11:
12: class QueensBoard {
13:     int size_;
14:     std::vector<int> board_;
15:
16: public:
17:     QueensBoard(int size)
18:             : size_(size), board_(size_, -1) {
19:     }
20:
21:     int size() const {
22:         return size_;
23:     }
24:
25:     bool under_attack(int row, int col) const {
26:         for (int i = 0; i < col; ++i) {
27:             if (board_[i] == -1) {
28:                 continue;
29:             }
30:             if (board_[i] == row) {
31:                 return true;
32:             }
33:             if (std::abs(i - col) == std::abs(board_[i] - row)) {
34:                 return true;
35:             }
36:         }
37:         return false;
38:     }
39:
40:     bool solve(int col = 0) {
41:         if (col == size()) {
42:             return true;
43:         }
44:         std::cout << "exploring col " << col << std::endl;
45:         for (int row = 0; row < size(); ++row) {
46:             if (!under_attack(row, col)) {
47:                 board_[col] = row;
48:                 std::cout << "placing queen on row " << row
49:                         << std::endl;
50:                 if (solve(col + 1)) {
51:                     return true;
52:                 }
53:             }
54:         }
55:         board_[col] = -1;
56:         return false;
57:     }
58:
59:     void pretty_print() const {
60:         std::cout << std::endl;
61:         for (int row = 0; row < size(); ++row) {
62:             for (int col = 0; col < size(); ++col) {
63:                 std::cout << '|';
64:                 if (board_[col] == row) {
65:                     std::cout << '*';
66:                 } else {
67:                     std::cout << ' ';
68:                 }
69:             }
70:             std::cout << '|' << std::endl;
71:         }
72:     }
73: };
74:
75: int main() {
76:
77:     QueensBoard qb(4);
78:     bool has_solution = qb.solve();
79:     std::cout << "has_solution=" << has_solution << std::endl;
80:     qb.pretty_print();
81:
82:     return 0;
83: }
84:
```

```
#include <iostream>
using namespace std;

void b(int, int);
void c(int, int);
void d(int, int);

void a(int i, int h) {

        if(i<=0) {
                return;
        }
        d(i-1, h);
        cout << '-' << h << ' ' << 0 << ' ' << "rlineto" << endl;
        a(i-1, h);
        cout << 0 << ' ' << '-' << h << ' ' << "rlineto" << endl;
        a(i-1, h);
        cout << h << ' ' << 0 << ' ' << "rlineto" << endl;
        b(i-1, h);

}

void b(int i, int h) {
        if(i<=0) {
                return;
        }
        c(i-1, h);
        cout << 0 << ' ' << h << ' ' << "rlineto" << endl;
        b(i-1, h);
        cout << h << ' ' << 0 << ' ' << "rlineto" << endl;
        b(i-1, h);
        cout << 0 << ' ' << '-' << h << ' ' << "rlineto" << endl;
        a(i-1, h);
}

void c(int i, int h) {
        if(i<=0) {
                return;
        }
        b(i-1, h);
        cout << h << ' ' << 0 << ' ' << "rlineto" << endl;
        c(i-1, h);
        cout << 0 << ' ' << h << ' ' << "rlineto" << endl;
        c(i-1, h);
        cout << '-' << h << ' ' << 0 << ' ' << "rlineto" << endl;
        d(i-1, h);
}

void d(int i, int h) {
        if(i<=0) {
                return;
        }
        a(i-1, h);
        cout << 0 << ' ' << '-' << h << ' ' << "rlineto" << endl;
        d(i-1, h);
        cout << '-' << h << ' ' << 0 << ' ' << "rlineto" << endl;
        d(i-1, h);
        cout << 0 << ' ' << h << ' ' << "rlineto" << endl;
        c(i-1, h);
}


int main() {
        cout << "newpath" << endl;
        int h = 512;
        int x0 = 64 + h/2;
        int y0 = 64 + h/2;
        for(int i = 1; i<5; ++i) {
          h /=2;
          x0 += h/2;
          y0 += h/2;
          cout << x0 << " " << y0 << " moveto" << endl;
          a(i, h);
          cout << 2*(6 - i + 1)/6.0 << " setlinewidth" << endl;
          cout << "stroke" << endl;
        }
        cout << "showpage" << endl;

        return 0;
}
```

```
#include <iostream>
#include <vector>
#include <string>
#include <cstdlib>
using namespace std;

enum Direction {
        NONE = 0,
        UP = 1,           // 0001
        RIGHT = 1 << 1,   // 0010
        DOWN = 1 << 2,    // 0100
        LEFT = 1 << 3     // 1000
};

class Cell {
        static const int PS_SIZE = 25;

        unsigned int row_;
        unsigned int col_;
        unsigned int walls_;

        bool visited_;

        string draw_wall(bool has_wall) const {
                return has_wall? "rlineto": "rmoveto";
        }
public:
        Cell(unsigned int row, unsigned int col)
        : row_(row),
          col_(col),
          walls_(UP | RIGHT | DOWN | LEFT), // 1111
          visited_(false)
        {}

        bool is_visited() const {
                return visited_;
        }

        Cell& visit() {
                visited_=true;
                return *this;
        }

        unsigned int get_row() const {
                return row_;
        }

        unsigned int get_col() const {
                return col_;
        }

        Cell& set_row(unsigned int row) {
                row_=row;
                return *this;
        }

        Cell& set_col(unsigned int col) {
                col_=col;
                return *this;
        }

        //  1100 1100
        //  0001 1000
        //  0000 1000

        bool has_wall(Direction dir) const {
                return walls_ & dir;
        }

        //  1100 1100
        //  0001 1000
        //  1101 1100

        void set_wall(Direction dir) {
                // walls_ = walls_ | dir;
                walls_ |= dir;
        }

        //  1100 1100
        //  1000 0001
        //~ 0111 1110
        //  0100 1100

        void unset_wall(Direction dir) {
                // walls_ = walls_ & ~dir;
                walls_ &= ~dir;
        }

        void draw() const {
                cout << col_*PS_SIZE << ' ' << row_*PS_SIZE
                     << " moveto" << endl;
                cout << PS_SIZE << " " << 0 << " "
                        << draw_wall(has_wall(DOWN)) << endl;
                cout << 0 << " " << PS_SIZE << " "
                        << draw_wall(has_wall(RIGHT)) << endl;
                cout << -PS_SIZE << " " << 0 << " "
                         << draw_wall(has_wall(UP)) << endl;
                cout << 0 << " " << -PS_SIZE << " "
                         << draw_wall(has_wall(LEFT)) << endl;
        }
};

class BoardError {};

class Board {

        unsigned int width_;
        unsigned int height_;
        vector<Cell> board_;

public:
        Board(unsigned int width, unsigned int height)
        : width_(width),
          height_(height)
        {
                for(unsigned int row=0; row<height_; row++) {
                        for(unsigned int col=0; col<width_; col++) {
                                board_.push_back(Cell(row, col));
                        }
                }
        }

        Cell& at(unsigned int row, unsigned int col) {
                return board_[row*width_+col];
        }

        const Cell& at(unsigned int row, unsigned int col) const {
                return board_[row*width_+col];
        }

        void draw() const {
                cout << "newpath" << endl;
                for(unsigned int row=0;row<height_; ++row) {
                        for(unsigned int col=0; col<width_; ++ col) {
```

```
135:                                at(row, col).draw();
136:                        }
137:                }
138:                cout << "stroke" << endl;
139:                cout << "showpage" << endl;
140:        }
141:
142:        bool has_neighbour(unsigned int row, unsigned int col,
143:                                                Direction dir) const {
144:                if(row==0 && dir==DOWN) {
145:                        return false;
146:                }
147:                if(row==(height_-1) && dir==UP) {
148:                        return false;
149:                }
150:                if(col==0 && dir==LEFT) {
151:                        return false;
152:                }
153:                if(col==(width_-1) && dir==RIGHT) {
154:                        return false;
155:                }
156:                return true;
157:        }
158:
159:        Cell& get_neighbour(unsigned int row, unsigned int col,
160:                                                Direction dir) {
161:                if(!has_neighbour(row, col, dir)) {
162:                        cerr << "heighbour not found: (" << row << "," << col
163:                                << ")" << endl;
164:                        throw BoardError();
165:                }
166:                unsigned int nr=(dir==UP)?row+1:((dir==DOWN)?row-1:row);
167:                unsigned int nc=(dir==LEFT)?col-1:((dir==RIGHT)?col+1:col);
168:                return at(nr, nc);
169:        }
170:
171:        Cell& drill(Cell& cell, Direction dir) {
172:                Cell& n=get_neighbour(cell.get_row(), cell.get_col(), dir);
173:
174:                cell.unset_wall(dir);
175:                switch(dir) {
176:                case UP:
177:                        n.unset_wall(DOWN);
178:                        break;
179:                case DOWN:
180:                        n.unset_wall(UP);
181:                        break;
182:                case LEFT:
183:                        n.unset_wall(RIGHT);
184:                        break;
185:                case RIGHT:
186:                        n.unset_wall(LEFT);
187:                        break;
188:                default:
189:                        cerr << "ala bala" << endl;
190:                        throw BoardError();
191:                }
192:                return n;
193:        }
194:
195: private:
196:        const static Direction DIRECTIONS[];
197:        const static unsigned int NDIR = 4;
198: public:
199:
200:        bool has_unvisited_neighbour(unsigned row, unsigned col) {
201:                for(unsigned int d=0; d < NDIR; d++) {
202:                        Direction dir = DIRECTIONS[d];
203:                        if(has_neighbour(row, col, dir)) {
204:                                Cell& c=get_neighbour(row, col, dir);
205:                                if(!c.is_visited()) {
206:                                        return true;
207:                                }
208:                        }
209:                }
210:                return false;
211:        }
212:
213:        Direction get_random_unvisited_neighbour(unsigned row,
214:
unsigned col) {
215:                if(!has_unvisited_neighbour(row, col)) {
216:                        throw BoardError();
217:                }
218:                while(true) {
219:                        unsigned d=rand()%NDIR;
220:                        Direction dir = DIRECTIONS[d];
221:                        if(has_neighbour(row, col, dir)) {
222:                                Cell& c=get_neighbour(row, col,dir);
223:                                if(!c.is_visited()) {
224:                                        return dir;
225:                                }
226:                        }
227:                }
228:        }
229:
230:        void generate_maze(unsigned row, unsigned col) {
231:                Cell& c = at(row, col);
232:                c.visit();
233:                while(true) {
234:                        if(!has_unvisited_neighbour(row, col)) {
235:                                return;
236:                        }
237:                        Direction dir=get_random_unvisited_neighbour(row, col);
238:                        Cell& n=drill(c, dir);
239:                        generate_maze(n.get_row(),
240:                                                n.get_col());
241:                }
242:        }
243: };
244:
245: const Direction Board::DIRECTIONS[]={DOWN, UP, LEFT, RIGHT};
246: //const Direction DIRECTIONS[]
247:
248: int main() {
249: /*
250:        Cell c1(0,1);
251:
252:        cout << c1.has_wall(UP) << endl;
253:        c1.unset_wall(UP);
254:        cout << c1.has_wall(UP) << endl;
255:        c1.set_wall(UP);
256:        cout << c1.has_wall(UP) << endl;
257: */
258:        Board b(20,20);
259:
260:        b.generate_maze(0, 10);
261:
262:        b.draw();
263:        /*
264:        cout << b.has_neighbour(0,1,DOWN)
265:             << b.has_neighbour(0,1,UP)
266:             << b.has_neighbour(0,1,LEFT)
267:             << b.has_neighbour(0,1,RIGHT) << endl;
```

```
268:         cout << b.has_neighbour(0,0,DOWN)
269:                  << b.has_neighbour(0,0,LEFT)
270:                  << b.has_neighbour(0,0,RIGHT)
271:                  << b.has_neighbour(0,0,UP) << endl;
272:         */
273:         return 0;
274: }
275:
276:
277:
```

```
#include <iostream>
#include <vector>
#include <cstdlib>
using namespace std;

enum Direction {
        NONE = 0,
        UP = 1 << 0,
        LEFT = 1 << 1,
        DOWN = 1 << 2,
        RIGHT = 1 << 3
};


class Cell {
        static const int WALL_SIZE = 20;

        unsigned int walls_;
        unsigned int row_;
        unsigned int col_;
        bool visited_;

        string draw_wall(bool has_wall) const {
                return has_wall?" rlineto":" rmoveto";
        }
public:
        Cell(unsigned int row, unsigned int col,
                unsigned int walls=UP|LEFT|DOWN|RIGHT)
        : walls_(walls),
          row_(row),
          col_(col),
          visited_(false)
        {}

        Cell& visit() {
                visited_=true;
                return *this;
        }

        bool is_visited() const {
                return visited_;
        }

        bool has_wall(Direction dir) const {
                return dir & walls_;
        }

        Cell& set_wall(Direction dir) {
                walls_ |= dir;
                return *this;
        }

        Cell& unset_wall(Direction dir) {
                walls_ &= ~dir;
                return *this;
        }

        unsigned get_row() const {
                return row_;
        }

        unsigned get_col() const {
                return col_;
        }

        void draw(ostream& out) const {
                out << (get_col()+1)*WALL_SIZE << ' '
                        << (get_row()+1)*WALL_SIZE << ' '
                        << "moveto" << endl;


                out << WALL_SIZE << ' ' << 0
                        << draw_wall(has_wall(DOWN)) << endl;
                out << 0 << ' ' << WALL_SIZE
                        << draw_wall(has_wall(RIGHT)) << endl;
                out << -WALL_SIZE << ' ' << 0
                        << draw_wall(has_wall(UP)) << endl;
                out << 0 << ' ' << -WALL_SIZE
                        << draw_wall(has_wall(LEFT)) << endl;
        }
};

class BoardError{};

class Board {
        unsigned width_;
        unsigned height_;
        vector<Cell> cells_;
public:
        Board(unsigned width, unsigned height)
        : width_(width),
          height_(height)
        {
                for(unsigned row=0;row<height_;row++) {
                        for(unsigned col=0;col<width_;col++) {
                                cells_.push_back(Cell(row,col));
                        }
                }
        }
        const Cell& get_cell(unsigned row, unsigned col) const {
                return cells_[row*width_+col];
        }

        Cell& get_cell(unsigned row, unsigned col) {
                return cells_[row*width_+col];
        }


        void draw(ostream& out) const {
                out << "newpath" << endl;
                for(vector<Cell>::const_iterator it=cells_.begin();
                        it!=cells_.end(); ++it) {

                        (*it).draw(out);
                }

                out << "stroke" << endl;
                out << "showpage" << endl;
        }

        bool has_neighbour(unsigned row, unsigned col,
                           Direction dir) const {
                if(row==0 && dir==DOWN)
                        return false;
                if(row==height_-1 && dir==UP)
                        return false;
                if(col==0 && dir==LEFT)
                        return false;
                if(col==width_-1 && dir==RIGHT)
                        return false;

                return true;
        }

```

```
135:         Cell& get_neighbour(unsigned row, unsigned col,
136:                             Direction dir) {
137:                 if(! has_neighbour(row, col, dir)) {
138:                         throw BoardError();
139:                 }
140:                 unsigned nr= (dir==UP)? row+1:(
141:                                 (dir == DOWN)? row-1: row);
142:                 unsigned nc= (dir==RIGHT) ? col+1: (
143:                                 (dir == LEFT)? col-1:col);
144:                 return get_cell(nr, nc);
145:         }
146:
147:         const Cell& get_neighbour(unsigned row, unsigned col,
148:                                                         Direction dir) const {
149:                 if(! has_neighbour(row, col, dir)) {
150:                         throw BoardError();
151:                 }
152:                 unsigned nr= (dir==UP)? row+1:(
153:                                 (dir == DOWN)? row-1: row);
154:                 unsigned nc= (dir==RIGHT) ? col+1: (
155:                                 (dir == LEFT)? col-1:col);
156:                 return get_cell(nr, nc);

157:         }
158:
159:         void drill_wall(unsigned row, unsigned col,
160:                                         Direction dir) {
161:                 Cell& c=get_cell(row, col);
162:                 c.unset_wall(dir);
163:
164:                 if(! has_neighbour(row, col, dir) ) {
165:                         return;
166:                 }
167:                 Cell& n=get_neighbour(row, col, dir);
168:                 Direction ndir;
169:
170:                 switch(dir) {
171:                 case UP:
172:                         n.unset_wall(DOWN);
173:                         break;
174:                 case RIGHT:
175:                         n.unset_wall(LEFT);
176:                         break;
177:                 case DOWN:
178:                         n.unset_wall(UP);
179:                         break;
180:                 case LEFT:
181:                         n.unset_wall(RIGHT);
182:                         break;
183:                 default:
184:                         throw BoardError();
185:                 }
186:         }
187:
188: private:
189:         const static Direction DIRECTIONS[];
190:         const static int DSIZE = 4;
191: public:
192:
193:         Direction has_unvisited_neighbour(int row,
194:                                                                         int col)
 const {
195:
196:                 for(int i=0;i<DSIZE;++i) {
197:                         Direction d=DIRECTIONS[i];
198:                         if(has_neighbour(row, col, d)) {
199:                                 const Cell& c=get_neighbour(row, col, d);
200:                                 if(!c.is_visited()) {
201:                                         return d;
202:                                 }
203:                         }
204:                 }
205:                 return NONE;
206:         }
207:
208:         Direction get_random_unvisited_neighbour(int row,
209:
int col) const {
210:                 if(!has_unvisited_neighbour(row, col)) {
211:                         return NONE;
212:                 }
213:                 while(true) {
214:                         int ind=rand()%DSIZE;
215:                         Direction d=DIRECTIONS[ind];
216:                         if(has_neighbour(row,col,d)) {
217:                                 const Cell& c=get_neighbour(row, col, d);
218:                                 if(!c.is_visited()) {
219:                                         return d;
220:                                 }
221:                         }
222:                 }
223:         }
224:
225:
226:         void generate(int row, int col) {
227:                 Cell& c=get_cell(row, col);
228:                 c.visit();
229:
230:                 while(true) {
231:                         Direction dir
232:                                 =get_random_unvisited_neighbour(row,col);
233:                         if(dir==NONE) {
234:                                 return;
235:                         }
236:                         drill_wall(row, col, dir);
237:                         Cell& n=get_neighbour(row, col, dir);
238:                         generate(n.get_row(), n.get_col());
239:                 }
240:
241:         }
242:
243: };
244:
245: const Direction Board::DIRECTIONS[] = {UP,LEFT,DOWN,RIGHT};
246:
247: int main() {
248:
249:         Board b(20, 20);
250:
251:         b.generate(0,0);
252:
253:         b.draw(cout);
254:
255:
256:
257:         return 0;
258: }
259:
260:
261:
262:
263:
264:
```

```
 1: /*
 2:  * maze.cc
 3:  *
 4:  *  Created on: Feb 22, 2017
 5:  *      Author: lubo
 6:  */
 7: #include <vector>
 8: #include <iostream>
 9:
10: enum Direction {
11:     NONE = 0, UP = 1 << 0, LEFT = 1 << 1, DOWN = 1 << 2, RIGHT = 1
12:             << 3
13: };
14:
15: class Cell {
16:     unsigned int walls_;
17:     unsigned int row_;
18:     unsigned int col_;
19:
20: public:
21:     Cell(unsigned int row, unsigned int col,
22:             unsigned walls = UP | LEFT | DOWN | RIGHT)
23:             : walls_(walls), row_(row), col_(col) {
24:     }
25:
26:     bool has_wall(Direction dir) const {
27:         return dir & walls_;
28:     }
29:
30:     Cell& set_wall(Direction dir) {
31:         walls_ |= dir;
32:         return *this;
33:     }
34:
35:     Cell& unset_wall(Direction dir) {
36:         walls_ &= ~dir;
37:         return *this;
38:     }
39:
40:     unsigned row() const {
41:         return row_;
42:     }
43:     unsigned col() const {
44:         return col_;
45:     }
46:
47: };
48:
49: class BoardError {
50: };
51:
52: class Board {
53:     unsigned width_;
54:     unsigned height_;
55:     std::vector<Cell> cells_;
56:
57:     unsigned index(unsigned row, unsigned col) const {
58:         if (row >= height() || col >= width()) {
59:             throw BoardError();
60:         }
61:         return row * col;
62:     }
63:
64:     unsigned nindex(unsigned row, unsigned col, Direction dir) const {
65:         int nrow =
66:                 dir == UP ? row + 1 : (dir == DOWN ? row - 1 : row);
67:         int ncol =
68:                 dir == LEFT ?
69:                         col - 1 : (dir == RIGHT ? col + 1 : col);
70:         if (nrow < 0 || nrow >= height() || ncol < 0
71:                 || ncol >= width()) {
72:             throw BoardError();
73:         }
74:         return index((unsigned) nrow, (unsigned) ncol);
75:     }
76: public:
77:
78:     Board(unsigned w, unsigned h)
79:             : width_(w), height_(h) {
80:         for (unsigned row = 0; row < height(); ++row) {
81:             for (unsigned col = 0; col < width(); ++col) {
82:                 cells_.push_back(Cell(row, col));
83:             }
84:         }
85:     }
86:
87:     unsigned width() const {
88:         return width_;
89:     }
90:     unsigned height() const {
91:         return height_;
92:     }
93:
94:     Cell& at(unsigned row, unsigned col) {
95:         return cells_[index(row, col)];
96:     }
97:
98:     const Cell& at(unsigned row, unsigned col) const {
99:         return cells_[index(row, col)];
100:    }
101:
102:    Cell& neighbour(unsigned row, unsigned col, Direction dir) {
103:        return cells_[nindex(row, col, dir)];
104:    }
105:
106:    const Cell& neighbour(unsigned row, unsigned col,
107:            Direction dir) const {
108:        return cells_[nindex(row, col, dir)];
109:    }
110:
111:    Cell& neighbour(const Cell& cell, Direction dir) {
112:        return cells_[nindex(cell.row(), cell.col(), dir)];
113:    }
114:
115:    const Cell& neighbour(const Cell& cell, Direction dir) const {
116:        return cells_[nindex(cell.row(), cell.col(), dir)];
117:    }
118:
119:    static Direction oposite_direction(Direction dir) {
120:        switch (dir) {
121:        case UP:
122:            return DOWN;
123:        case DOWN:
124:            return UP;
125:        case LEFT:
126:            return RIGHT;
127:        case RIGHT:
128:            return LEFT;
129:        default:
130:            throw BoardError();
131:        }
132:    }
133:
134:    Cell& drill(Cell& cell, Direction dir) {
```

```
135:         cell.unset_wall(dir);
136:         Cell& ncell = neighbour(cell, dir);
137:         Direction oposite = oposite_direction(dir);
138:         ncell.unset_wall(oposite);
139:         return ncell;
140:     }
141:
142: };
143:
144: int main() {
145:     Board b(10, 10);
146:
147:     return 0;
148: }
```

```
    1: CXXFLAGS = -g -Wall
    2:
    3:
    4: OBJ = queens.o
    5: SRC = queens.cc hilbert.cc maze01.cc maze02.cc maze.cc
    6:
    7: OUT = queens
    8:
    9:
   10: all: $(OUT)
   11:
   12:
   13: $(OUT): $(OBJ)
   14:         g++ $(CXXFLAGS) $(OBJ) -o $(OUT)
   15:
   16:
   17: clean:
   18:         rm -f *~ a.out $(OUT) *.o files.pdf
   19:
   20:
   21: files.pdf: $(SRC)
   22:         enscript -r -2 --highlight --line-numbers -o - $(SRC) Makefile | ps2pdf -
files.pdf
   23:
   24: pdf: files.pdf
```