

List 1: main.cpp

```

1  #include <iostream>
2  #include <vector>
3  // for OpenCV2
4  #include <opencv2/core/core.hpp>
5  #include <opencv2/highgui/highgui.hpp>
6
7  // httpクライアント
8  #include "httpClient/client.h"
9  // 問題ヘッダ部(コメント部分)
10 #include "httpClient/QuestionHeader.hpp"
11
12 // 画像解析
13 #include "placement/PPMFILE.hpp"
14
15 // sort
16 #include "sort/PosData.h"
17 #include "sort/Process5.h"
18
19 // デバッグ用
20 #define VERBOSE
21
22 using namespace std;
23
24 // 問題解答関数
25 int solveProbrem(int id);
26
27 int main(){
28     int id=1;
29
30     // マルチプロセスとかしたい時のために
31     solveProbrem(id);
32
33     return EXIT_SUCCESS;
34 }
35
36 int solveProbrem(int id){
37     ProkonClient client; // httpクライアント
38     QuestionHeader header; // ヘッダデータ
39     string res; // 受信データ
40     string answer; // 解答データ
41     // 受信した画像データ
42     cv::Mat recievedData;
43     // PPMFILE
44     PPMFILE *img;
45     // PosData
46     PosData *data;
47     Process5 *sort;
48
49 #ifdef VERBOSE
50     cout << "Picture Downloading\nID: " << id<< endl;
51 #endif
52
53     try {
54         // 引数に問題番号を指定して画像のバイナリ stringを返す(cv::Mat形式にするかも)
55         res=client.getProblem(id,header);
56     } catch (runtime_error & exception) {
57         cerr << "Exception: " << exception.what() << endl;
58         exit(EXIT_FAILURE);
59     }
60
61 #ifdef VERBOSE
62     cerr << "Split X: " << header.splitX << endl;
63     cerr << "Split Y: " << header.splitY << endl;
64     cerr << "Selectable Count: " << header.selectableCount << endl;
65     cerr << "Select Rate: " << header.selectRate << endl;
66     cerr << "Exchange Rate: " << header.exchangeRate << endl;

```

```

67 #endif
68
69 // Matにするために受け取った文字列をvector<char>型に変換
70 vector<char> v(res.begin(),res.end());
71 recievedData=cv::imdecode(cv::Mat(v),-1);
72
73 data=new PosData(header.splitX,header.splitY);
74 sort=new Process5(header.splitX,header.splitY);
75
76 #ifdef VERBOSE
77     cout << "placement" << endl;
78 #endif
79 // ここに画像解析処理
80 // どこかでPosData設定が必要
81 img= new PPMFILE(recievedData,header.splitX,header.splitY);
82 img->calc_cost();
83 // img->calc_cost_maru(); // まだこのコスト計算は実装されていない
84 img->placement();
85 // img->create_result_img(); //画像を表示しないなら使わないでOK
86 img->set_PosData(data);
87
88 #ifdef VERBOSE
89     cout << "sort" << endl;
90 #endif
91 // ここにソート処理
92 sort->importData(*data);
93
94 #ifdef VERBOSE
95     cout << "(not sort done)Answer And Send" << endl;
96 #endif
97
98 try {
99     // 引数に問題番号を指定して画像のバイナリstringを返す(cv::Mat形式にするかも)
100     res=client.sendAnswer(id,sort->sort());
101 } catch (char const * exception) {
102
103     cerr << "Exception: " << exception << endl;
104     exit(EXIT_FAILURE);
105 }
106 cout << res << endl;
107
108 return EXIT_SUCCESS;
109
110 }

```

List 2: httpClient/QuestionHeader.hpp

```

1 #ifndef __QUESTION_HEADER_H_INCLUDED__
2 #define __QUESTION_HEADER_H_INCLUDED__
3 // 問題ヘッダクラス
4 // ヘッダは分割数 選択可能回数 コスト変換レートが書かれている
5 // 画像側は分割数, ソート側は選択可能回数を使えばいいと思う
6 class QuestionHeader {
7     public:
8         // 分割数X,Y
9         unsigned int splitX;
10        unsigned int splitY;
11        // レート(選択, 交換)
12        unsigned int selectRate;
13        unsigned int exchangeRate;
14        // 交換可能回数
15        unsigned int selectableCount;
16 };
17
18 #endif

```

List 3: httpClient/client.cpp

```

1  #include <curl/curl.h>
2  #include <iostream>
3  #include <sstream>
4  #include <string>
5  #include <vector>
6  // for isdigit
7  #include <cctype>
8
9  #include "client.h"
10 #include "QuestionHeader.hpp"
11
12 #define DEBUG 1
13
14 using namespace std;
15
16 // サーバー
17 const string ProkonClient::SERVER_ADDRESS="localhost/web2/pic";
18 // チーム固有トークン
19 const string ProkonClient::TEAM_TOKEN="SKYHIGH\nCHRONOS\nENDLESS\n";
20
21 // コンストラクタ
22 ProkonClient::ProkonClient(){
23     curl = curl_easy_init();
24     if(curl==NULL){
25         throw runtime_error("CURL is NULL\n");
26     }
27 }
28 // デストラクタ
29 ProkonClient::~ProkonClient(){
30     curl_easy_cleanup(curl);
31 }
32
33 string ProkonClient::getProblem(int problemNo, QuestionHeader & data){
34     // 受信データ
35     string chunk;
36     // 入れるポインタデータ
37     vector<unsigned int *> dataPointer;
38     // 現在操作するポインタ
39     vector<unsigned int *>::iterator now;
40     // フラグ
41     bool enable;
42
43     // 代入するデータを順に入れる
44     dataPointer.push_back(& data.splitX);
45     dataPointer.push_back(& data.splitY);
46     dataPointer.push_back(& data.selectableCount);
47     dataPointer.push_back(& data.selectRate);
48     dataPointer.push_back(& data.exchangeRate);
49
50     // URL用問題番号の文字列
51     ostringstream probStr;
52     //問題番号部分指定
53     probStr.setf(ios::right);
54     probStr.fill('0');
55     probStr.width(2);
56     probStr << problemNo;
57
58     // URL
59     string url="http://" + SERVER_ADDRESS + "/problem/prob" + probStr.str() + ".ppm"
60     ;
61     #if DEBUG==1
62         cout << url << endl;
63     #endif
64
65     chunk=getData(url);
66     // コメント抜き取り

```

```

66 now=dataPointer.begin();
67 **now=0;
68 enable=false;
69 for(char &c : chunk){
70     // コメント読み取り有効
71     if(c=='#'){
72         // 有効
73         enable=true;
74     }
75     // コメント読み取り無効
76     if(c=='\n'){
77         enable=false;
78     }
79     if(enable){
80         if(isdigit(c)){
81             **now= (**now) * 10+(c-'0');
82         }else if(**now!=0){
83             now++;
84         }
85     }
86     if(now == dataPointer.end()){
87         break;
88     }
89 }
90 return chunk;
91 }
92
93 string ProkonClient::sendAnswer(int problemNo,string answer){
94     // 送信データ
95     string data;
96     // 受信データ
97     string chunk;
98     //http://{ServerAddress}/SubmitAnswer
99     string url="http://" + SERVER_ADDRESS + "/SubmitAnswer.pl";
100     // エスケープされた文字列 free用
101     char * escapedStr;
102
103     data="playerid=";
104     // チームトークン
105     escapedStr=curl_easy_escape(curl,TEAM_TOKEN.c_str(),0);
106     data+=escapedStr;
107     curl_free(escapedStr);
108     // 問題番号
109     escapedStr=curl_easy_escape(curl,to_string(problemNo).c_str(),0);
110     data+="&problemid="+string(escapedStr);
111     curl_free(escapedStr);
112     //回答
113     escapedStr=curl_easy_escape(curl,answer.c_str(),0);
114     data+="&answer="+string(escapedStr);
115     curl_free(escapedStr);
116
117     chunk=getData(url,data);
118
119     return chunk;
120 }
121
122 // postData: *直接送信する* (エスケープが必要)
123 string ProkonClient::getData(string url,string postData){
124     CURLcode res;
125     // httpコード (200とか404とか)
126     long http_code=0;
127     // ダウンロードしたデータ
128     string chunk;
129
130     // URL設定
131     curl_easy_setopt(curl,CURLOPT_URL,url.c_str());
132     // port:8080

```

```

133 curl_easy_setopt(curl,CURLOPT_PORT,8080);
134 // POSTデータ指定
135 curl_easy_setopt(curl,CURLOPT_POSTFIELDS,postData.c_str());
136
137 // 受信コールバック
138 curl_easy_setopt(curl,CURLOPT_WRITEFUNCTION,ProkonClient::callbackWrite);
139 // コールバックデータ書き込み先
140 curl_easy_setopt(curl,CURLOPT_WRITEDATA,&chunk);
141 // 実行
142 res=curl_easy_perform(curl);
143 // レスponseコード受信
144 curl_easy_getinfo(curl,CURLINFO_RESPONSE_CODE,&http_code);
145
146 if(res != CURLE_OK){
147     cerr << "!!!curl_easy_perform failed!!!" << endl;
148     throw runtime_error("perform something wrong.");
149 }
150 if(http_code != 200){
151     cerr << "STATUS CODE IS NOT 200:" << to_string(http_code) << endl;
152     string msg="http code is "+to_string(http_code);
153     throw runtime_error(msg);
154 }
155 return chunk;
156 }
157
158 string ProkonClient::getData(string url){
159     return getData(url,"");
160 }
161
162 size_t ProkonClient::callbackWrite(char *ptr,size_t size, size_t nmemb,string *
    stream){
163     // サイズ計算
164     int dataLength = size * nmemb;
165     // 書き込み
166     stream->append(ptr,dataLength);
167     // サイズを返す
168     return dataLength;
169 }

```

List 4: httpClient/client.h

```

1  #ifndef __PROKON_CLIENT_H_INCLUDED__
2  #define __PROKON_CLIENT_H_INCLUDED__
3
4  #include <string>
5  #include <stdio.h>
6  #include <fstream>
7  #include <curl/curl.h>
8
9  #include "QuestionHeader.hpp"
10
11 using namespace std;
12
13 class ProkonClient{
14     private:
15         // streamにsize*nmembの大きさのptrを追加
16         // 大きさを返す
17         static size_t callbackWrite(char *ptr,size_t size, size_t nmemb,string *stream
            );
18         string getData(string url,string postData);
19         string getData(string url);
20         // curlオブジェクト
21         CURL *curl;
22     public:
23         // コンストラクタ
24         ProkonClient();

```

```

25 ~ProkonClient();
26 static const string SERVER_ADDRESS;
27 static const string TEAM_TOKEN;
28 string getProblem(int problemNo, QuestionHeader & data);
29 string sendAnswer(int problemNo, string answer);
30 };
31 #endif

```

List 5: placement/PPMFILE.cpp

```

1  #include <iostream>
2
3  #include <opencv2/core/core.hpp>
4  #include <opencv2/highgui/highgui.hpp>
5
6  #include "PPMFILE.hpp"
7  #include "../sort/PosData.h"
8
9  // tupleを比較するときのルール
10 bool my_compare( const COST_TUPLE &lhs, const COST_TUPLE &rhs){
11     if (std::get<0>(lhs) != std::get<0>(rhs)) return std::get<0>(lhs) < std::
        get<0>(rhs);
12     if (std::get<1>(lhs) != std::get<1>(rhs)) return std::get<1>(lhs) < std::
        get<1>(rhs);
13     if (std::get<2>(lhs) != std::get<2>(rhs)) return std::get<2>(lhs) < std::
        get<2>(rhs);
14     return std::get<2>(lhs) < std::get<2>(rhs);
15 }
16
17 // 方向に対するpairを作る
18 pair<int,int> make_direction_pair(int direction){
19     pair<int,int> dire_pair;
20     switch(direction){
21         case DIRE_U:
22             dire_pair = make_pair(0,-1);
23             break;
24         case DIRE_D:
25             dire_pair = make_pair(0,1);
26             break;
27         case DIRE_R:
28             dire_pair = make_pair(1,0);
29             break;
30         case DIRE_L:
31             dire_pair = make_pair(-1,0);
32             break;
33     }
34
35     return dire_pair;
36 }
37
38 // 方向を逆にする
39 int inverse_direction(int dire){
40     int inv_dire;
41     switch(dire){
42         case DIRE_U:
43             inv_dire = DIRE_D;
44             break;
45         case DIRE_D:
46             inv_dire = DIRE_U;
47             break;
48         case DIRE_R:
49             inv_dire = DIRE_L;
50             break;
51         case DIRE_L:
52             inv_dire = DIRE_R;
53             break;

```

```

54         default:
55             cout << "error direction" << endl;
56             break;
57     }
58     return inv_dire;
59 }
60
61 PPMFILE::PPMFILE(cv::Mat origin_img_tmp, int piece_x, int piece_y){
62     origin_img = origin_img_tmp.clone();
63     part_size_x = piece_x;
64     part_size_y = piece_y;
65
66     //それぞれのピースに分割
67     this->create_partition();
68 }
69
70 void PPMFILE::disp_img(int type){
71     cv::namedWindow("image", CV_WINDOW_AUTOSIZE | CV_WINDOW_FREERATIO);
72     switch (type){
73         case ORIGIN_IMG:         cv::imshow("image", origin_img);
74                                 break;
75         case LINE_IMG: cv::imshow("image", line_img);
76                                 break;
77         case RESULT_IMG:         cv::imshow("image", result_img);
78                                 break;
79         default:
80                                 break;
81     }
82 }
83
84 void PPMFILE::write_line(void){
85     line_img = origin_img.clone(); // origin_img をコピー
86     //縦向きの線
87     for(int x=1; x < part_size_x; x++){
88         cv::line( line_img, cv::Point( x*(line_img.cols/part_size_x), 0),
89                 cv::Point( x*(line_img.cols/part_size_x), line_img.rows), cv
90                 ::Scalar( 200, 0, 0), 2, 0);
91
92     }
93     //横向きの線
94     for(int y=1; y < part_size_y; y++){
95         cv::line( line_img, cv::Point( 0, y*(line_img.rows/part_size_y)),
96                 cv::Point( line_img.cols, y*(line_img.rows/part_size_y)), cv
97                 ::Scalar( 0, 0, 200), 2, 0);
98     }
99 }
100
101 void PPMFILE::create_partition(void){
102     int part_width = origin_img.cols/part_size_x;
103     int part_height = origin_img.rows/part_size_y;
104     //pair_img[n]にそれぞれのピースを代入
105     for(int y=0; y < part_size_y; y++){
106         for(int x=0; x < part_size_x; x++){
107             cv::Mat tmp(origin_img, cv::Rect(x*part_width, y*
108             part_height, part_width, part_height));
109             part_img.push_back(tmp);
110         }
111     }
112 }
113
114 void PPMFILE::calc_cost(void){
115     // costのサイズ変更
116     cost.resize( part_size_x*part_size_y);
117     for(int i=0; i<part_size_x*part_size_y; i++){
118         cost[i].resize(4);
119         for(int j=0; j<4; j++){
120             cost[i][j].resize(part_size_x*part_size_y);
121         }
122     }
123 }

```

```

116     }
117     cout << "start_calc" << endl;
118     int cost_tmp[4];
119     // 差分積算量は数が少ない方から多い方にじゃないと調べられない
120     for(int abs_xy=0; abs_xy < part_size_x * part_size_y; ++abs_xy){
121         for(int xy=abs_xy+1; xy < part_size_x * part_size_y; ++xy){
122             //コスト初期化
123             for(int i=0; i<4; i++){
124                 cost_tmp[i] = 0;
125             }
126             //カラーの数だけ繰り返す
127             for(int c=0; c < part_img[abs_xy].channels(); ++c){
128                 //各ピクセル上下
129                 for(int px_x=0; px_x < part_img[0].cols; ++px_x){
130                     cost_tmp[DIRE_U] += abs(part_img[abs_xy].
                        at<cv::Vec3b>( 0, px_x)[c] - part_img[
                        xy].at<cv::Vec3b>( part_img[0].rows-1,
                        px_x)[c]);
131                     cost_tmp[DIRE_D] += abs(part_img[abs_xy].
                        at<cv::Vec3b>( part_img[0].rows-1,
                        px_x)[c] - part_img[xy].at<cv::Vec3b>(
                        0, px_x)[c]);
132                 }
133                 //各ピクセル左右
134                 for(int px_y=0; px_y < part_img[0].rows; ++px_y){
135                     cost_tmp[DIRE_R] += abs(part_img[abs_xy].
                        at<cv::Vec3b>( px_y, part_img[0].cols
                        -1)[c] - part_img[xy].at<cv::Vec3b>(
                        px_y, 0)[c]);
136                     cost_tmp[DIRE_L] += abs(part_img[abs_xy].
                        at<cv::Vec3b>( px_y, 0)[c] - part_img[
                        xy].at<cv::Vec3b>( px_y, part_img[0].
                        cols-1)[c]);
137                 }
138             }
139             // 上下に対しては縦のピクセル数、左右に対しては横のピクセル数
140             // を×る事によって、結合度の重みを長さに依存させない
141             // ワンちゃんオーバーフローが怖い(たぶん大丈夫)
142             cost_tmp[DIRE_U] *= part_img[0].rows;
143             cost_tmp[DIRE_D] *= part_img[0].rows;
144             cost_tmp[DIRE_R] *= part_img[0].cols;
145             cost_tmp[DIRE_L] *= part_img[0].cols;
146             // cost_t [ コスト, 自分の座標, 相手の座標, 方向]
147             // cost [自分の座標][方向][相手の座標]
148             for(int k=0; k < 4; ++k){
149                 cost_t.push_back(make_tuple( cost_tmp[k], abs_xy,
                        xy, k));
150                 cost[abs_xy][k][xy] = make_pair( cost_tmp[k], xy
                        );
151             }
152         }
153     }
154     cout << "start_sort" << endl;
155     // 独自のルールで COST_TUPLEをソート
156     sort( cost_t.begin(), cost_t.end(), my_compare);
157     cout << "end calc" << endl;
158 }
159
160 void PPMFILE::disp_cost_list(void){
161     // 表示
162     cout << "score : my_pos : pair_pos : direction " << endl;
163     for(int i=0; i < cost_t.size(); i++){
164         cout << i << " |cost: " << get<0>(cost_t[i]) << ", (" << CONV_X(
            get<1>(cost_t[i])) << ", " << CONV_Y( get<1>(cost_t[i])) << "),
            (" << CONV_X(get<2>(cost_t[i])) << ", " << CONV_Y(get<2>(
            cost_t[i])) << ")", " << get<3>(cost_t[i]) << ", " << endl;

```



```

165     }
166 }
167
168 void PPMFILE::placement(void){
169     // 使われたかどうかのフラグ
170     vector<SCRAP> scraps;
171     vector<int> used_part(part_size_x*part_size_y, -1);
172     // 目的の場所への差分
173     int dif_x, dif_y;
174     int part_s, part_l;
175     int part_s_x, part_s_y;
176     int part_l_x, part_l_y;
177     int part_1, part_2;
178     int part_1_x, part_1_y;
179     int part_2_x, part_2_y;
180     // 再配置の時に使う
181     int pos_x_min = 0;
182     int pos_y_min = 0;
183     for(int i=0; i < cost_t.size(); i++){
184         SCRAP scrap_tmp;
185         part_1 = used_part[get<1>(cost_t[i])];
186         part_2 = used_part[get<2>(cost_t[i])];
187
188         if(part_1 == -1){
189             if(part_2 == -1){ // まだ使われてなかったら
190                 // どちらもまだ使われていない
191                 used_part[get<1>(cost_t[i])] = used_part[get<2>(
192                     cost_t[i])] = scraps.size(); // 追加した
193                     scrap番号を保持
194                 scrap_tmp.elements[get<1>(cost_t[i])] = make_pair
195                     ( 0, 0);
196                 scrap_tmp.used_p[make_pair(0,0)] = get<1>(cost_t[
197                     i]);
198                 scrap_tmp.elements[get<2>(cost_t[i])] =
199                     make_direction_pair(get<3>(cost_t[i]));
200                 scrap_tmp.used_p[make_direction_pair(get<3>(
201                     cost_t[i]))] = get<2>(cost_t[i]);
202                 scraps.push_back(scrap_tmp);
203             }
204             else{ // 2だけ
205                 // つかわれてる
206                 part_2_x = scraps[part_2].elements[get<2>(cost_t[
207                     i])].first;
208                 part_2_y = scraps[part_2].elements[get<2>(cost_t[
209                     i])].second;
210                 // 目的の場所(2が基準だから左右が逆になる)
211                 switch(get<3>(cost_t[i])){
212                     case DIRE_U:
213                         ++part_2_y;
214                         break;
215                     case DIRE_D:
216                         --part_2_y;
217                         break;
218                     case DIRE_R:
219                         --part_2_x;
220                         break;
221                     case DIRE_L:
222                         ++part_2_x;
223                         break;
224                 }
225                 if(scraps[part_2].used_p.find(make_pair(part_2_x,
226                     part_2_y)) == scraps[part_2].used_p.end()){
227                     used_part[get<1>(cost_t[i])] = part_2;
228                     scraps[part_2].elements[get<1>(cost_t[i]
229                         )]] = make_pair( part_2_x, part_2_y);
230                     scraps[part_2].used_p[make_pair( part_2_x
231                         , part_2_y)] = get<1>(cost_t[i]);

```

```

220         }else{
221             // すでに要素が入っていた場合
222         }
223     }
224 }else{
225     if(part_2 == -1){ // 1だけ使われてる
226         part_1_x = scraps[part_1].elements[get<1>(cost_t[
227             i])].first;
228         part_1_y = scraps[part_1].elements[get<1>(cost_t[
229             i])].second;
230         // 目的の場所
231         switch(get<3>(cost_t[i])){
232             case DIRE_U:
233                 --part_1_y;
234                 break;
235             case DIRE_D:
236                 ++part_1_y;
237                 break;
238             case DIRE_R:
239                 ++part_1_x;
240                 break;
241             case DIRE_L:
242                 --part_1_x;
243                 break;
244         }
245         // 目的の場所に要素が入っていなかった場合
246         if(scraps[part_1].used_p.find(make_pair(part_1_x,
247             part_1_y)) == scraps[part_1].used_p.end()){
248             used_part[get<2>(cost_t[i])] = part_1;
249             scraps[part_1].elements[get<2>(cost_t[i
250                 ])] = make_pair( part_1_x, part_1_y);
251             scraps[part_1].used_p[make_pair( part_1_x
252                 , part_1_y)] = get<2>(cost_t[i]);
253         }else{
254             // 既に要素が入っていた場合
255         }
256     }else{ // 両方ともつかわれている -> スクラップの添字が小
257         // さい方に大きい方をくっつける
258         if( part_1 != part_2){ // 同じスクラップでない場
259             合
260             part_s = MIN_2( part_1, part_2);
261             part_l = BIG_2( part_1, part_2);
262
263 #ifdef DEBUG
264             cout << "now data " << endl;
265             for(int i=0; i < scraps.size(); i++){
266                 cout << "pair : " << i << endl;
267                 for(map<int, pair<int,int> >::
268                     iterator j = scraps[i].
269                     elements.begin(); j != scraps[
270                         i].elements.end(); j++){
271                     int key = j->first;
272                     pair<int, int> pos = j->
273                         second;
274                     cout << " (" << CONV_X(
275                         key) << "," << CONV_Y(
276                         key) << ") || (" <<
277                         pos.first << "," <<
278                         pos.second << ")" <<
279                         endl;
280                 }
281             }
282             cout << "part_1 : " << part_1 << endl;
283             cout << "marg : " << part_s << " to " <<
284                 part_l << endl;
285             cout << "PP1 : (" << CONV_X(get<1>(cost_t
286                 [i])) << "," << CONV_Y(get<1>(cost_t[i

```

```

269     ))) << ")" << endl;
270     cout << "PP2 : (" << CONV_X(get<2>(cost_t
271     [i])) << "," << CONV_Y(get<2>(cost_t[i
272     ]))) << ")" << endl;
273     cout << "discription : " << get<3>(cost_t
274     [i]) << endl;
275
276 #endif
277
278     if(part_s == part_1){
279         cout << "type:A" << endl;
280         part_s_x = scraps[part_1].
281             elements[get<1>(cost_t[i])].
282             first;
283         part_s_y = scraps[part_1].
284             elements[get<1>(cost_t[i])].
285             second;
286         part_l_x = scraps[part_2].
287             elements[get<2>(cost_t[i])].
288             first;
289         part_l_y = scraps[part_2].
290             elements[get<2>(cost_t[i])].
291             second;
292         cout << "p_s_x : " << part_s_x <<
293             " p_s_y : " << part_s_y <<
294             endl;
295         switch(get<3>(cost_t[i])){
296             case DIRE_U:
297                 --part_s_y;
298                 break;
299             case DIRE_D:
300                 ++part_s_y;
301                 break;
302             case DIRE_R:
303                 ++part_s_x;
304                 break;
305             case DIRE_L:
306                 --part_s_x;
307                 break;
308         }
309     }else{
310         // part_2 を基準とする
311         cout << "type:B" << endl;
312         part_l_x = scraps[part_1].
313             elements[get<1>(cost_t[i])].
314             first;
315         part_l_y = scraps[part_1].
316             elements[get<1>(cost_t[i])].
317             second;
318         part_s_x = scraps[part_2].
319             elements[get<2>(cost_t[i])].
320             first;
321         part_s_y = scraps[part_2].
322             elements[get<2>(cost_t[i])].
323             second;
324         switch(get<3>(cost_t[i])){
325             case DIRE_U:
326                 ++part_s_y;
327                 break;
328             case DIRE_D:
329                 --part_s_y;
330                 break;
331             case DIRE_R:
332                 --part_s_x;
333                 break;
334             case DIRE_L:
335                 ++part_s_x;
336                 break;
337         }
338     }

```

```

314     }
315     cout << "p_s_x : " << part_s_x << " p_s_y
        : " << part_s_y << endl;
316     dif_x = part_s_x - part_l_x;
317     dif_y = part_s_y - part_l_y;
318     cout << "dif_x : " << dif_x << " dif_y :
        " << dif_y << endl;
319     // scrapが小さい方に統合(何も考えずに)
320     for(map<int, pair<int,int> >::iterator j
        = scraps[part_l].elements.begin(); j
        != scraps[part_l].elements.end(); j
        ++){
321         int key = j->first;
322         used_part[key] = part_s;
323         pair<int, int> pos = j->second;
324         scraps[part_s].elements[key] =
            make_pair( pos.first + dif_x,
325                     pos.second + dif_y);
            scraps[part_s].used_p[make_pair(
                pos.first + dif_x, pos.second
                + dif_y)] = key;
326     }
327     scraps[part_l].elements.clear();
328     for(int i=0; i < scraps.size(); i++){
329         cout << "pair : " << i << endl;
330         for(map<int, pair<int,int> >::
            iterator j = scraps[i].
            elements.begin(); j != scraps[
            i].elements.end(); j++){
331             int key = j->first;
332             pair<int, int> pos = j->
                second;
333             cout << " (" << CONV_X(
                key) << "," << CONV_Y(
                key) << ") || (" <<
                pos.first << "," <<
                pos.second << ")" <<
                endl;
334         }
335     }
336 }
337 }
338 }
339 }
340 // 座標の左上を0,0にする
341 // この時点でscrapsは[0]しか要素を持たないはず
342 for(map<int, pair<int,int> >::iterator j = scraps[0].elements.begin(); j
    != scraps[0].elements.end(); j++){
343     int key = j->first;
344     pair<int, int> pos = j->second;
345     if(pos.first < pos_x_min)
346         pos_x_min = pos.first;
347     if(pos.second < pos_y_min)
348         pos_y_min = pos.second;
349 }
350 // 一番小さいものに座標を合わせて再配置
351 for(map<int, pair<int,int> >::iterator j = scraps[0].elements.begin(); j
    != scraps[0].elements.end(); j++){
352     int key = j->first;
353     pair<int, int> pos = j->second;
354     scraps[0].elements[key] = make_pair( (pos.first - pos_x_min), (
        pos.second - pos_y_min));
355 }
356 // グローバルに結果をコピー
357 placement_pos = scraps[0].elements;
358 }
359

```

```

360 void PPMFILE::disp_placement(void){
361     cout << "-----" << endl;
362     cout << "placement " << endl;
363     for(map<int, pair<int,int> >::iterator j = placement_pos.begin(); j !=
        placement_pos.end(); j++){
364         int key = j->first;
365         pair<int, int> pos = j->second;
366         cout << " (" << CONV_X(key) << "," << CONV_Y(key) << ") || (" <<
            pos.first << "," << pos.second << ") " << endl;
367     }
368 }
369
370 void PPMFILE::create_result_img(void){
371     int part_width = origin_img.cols/part_size_x;
372     int part_height = origin_img.rows/part_size_y;
373
374     int max_part_x = 0, max_part_y = 0;
375
376     // 最大の幅と高さを求める
377     for(map<int, pair<int,int> >::iterator j = placement_pos.begin(); j !=
        placement_pos.end(); j++){
378         int key = j->first;
379         pair<int, int> pos = j->second;
380         if(max_part_x < pos.first)
381             max_part_x = pos.first;
382         if(max_part_y < pos.second)
383             max_part_y = pos.second;
384     }
385     cout << "mpx :" << max_part_x << " mpy :" << max_part_y << endl;
386
387     // 大きさに応じたサイズの result_img を作成
388     cv::Mat tmp_result_img( cv::Size((max_part_x+1) * part_width, (max_part_y
        +1) * part_height), CV_8UC3, cv::Scalar(0,0,0));
389     // cv::Mat tmp_result_img( cv::Size( 2000, 2000), CV_8UC3, cv::Scalar
        (0,0,0));
390
391     for(map<int, pair<int,int> >::iterator j = placement_pos.begin(); j !=
        placement_pos.end(); j++){
392         int key = j->first;
393         pair<int, int> pos = j->second;
394         part_img[key].copyTo(tmp_result_img( \
395             cv::Rect( \
396                 pos.first*part_width, \
397                 pos.second*part_height, \
398                 part_width, \
399                 part_height) \
400             ));
401         cout << "s_x :" << pos.first*part_width << " s_y :" << pos.
            second*part_height << " CopyTo:" << key << endl;
402     }
403     result_img = tmp_result_img.clone();
404 }
405
406 int PPMFILE::get_cost(int xy_1, int xy_2, int dire){
407     int pair_cost;
408     if(xy_1 < xy_2){
409         pair_cost = cost[xy_1][dire][xy_2].first;
410     }else if(xy_1 > xy_2){
411         // 比較順が変わると上下左右が変わる
412         dire = inverse_direction(dire);
413         pair_cost = cost[xy_2][dire][xy_1].first;
414     }else{
415         pair_cost = -1;
416     }
417     return pair_cost;
418 }
419

```

```

420 void PPMFILE::set_PosData(PosData *data){
421     for(map<int, pair<int,int> >::iterator j = placement_pos.begin(); j !=
422         placement_pos.end(); j++){
423         int key = j->first;
424         pair<int, int> pos = j->second;
425
426         data->setData( CONV_X(key), CONV_Y(key), pos.first, pos.second);
427     }
428 }
429
430 void PPMFILE::calc_cost_maru(void){
431     // costのサイズ変更
432     cost_maru.resize( part_size_x*part_size_y);
433     for(int i=0; i<part_size_x*part_size_y; i++){
434         cost_maru[i].resize(4);
435         for(int j=0; j<4; j++){
436             cost_maru[i][j].resize(part_size_x*part_size_y);
437         }
438     }
439
440     cout << "start_calc_maru" << endl;
441
442     int cost_tmp[4];
443     // 差分積算量は数が少ない方から多い方にしないと調べられない
444     for(int abs_xy=0; abs_xy < part_size_x * part_size_y; ++abs_xy){
445         for(int xy=abs_xy+1; xy < part_size_x * part_size_y; ++xy){
446             //コスト初期化
447             for(int i=0; i<4; i++){
448                 cost_tmp[i] = 0;
449             }
450             //カラーの数だけ繰り返す
451             for(int c=1; c < part_img[abs_xy].channels(); ++c){
452                 //各ピクセル上下
453                 for(int px_x=0; px_x < part_img[0].cols; ++px_x){
454                     cost_tmp[DIRE_U] += abs(part_img[abs_xy].
455                         at<cv::Vec3b>( 0, px_x)[c] - (part_img
456                             [abs_xy].at<cv::Vec3b>( 1, px_x)[c] -
457                             part_img[xy].at<cv::Vec3b>( part_img
458                                 [0].rows-1, px_x)[c])/2);
459                     cost_tmp[DIRE_D] += abs(part_img[abs_xy].
460                         at<cv::Vec3b>( part_img[0].rows-1,
461                             px_x)[c] - (part_img[abs_xy].at<cv::
462                                 Vec3b>( 0, px_x)[c] - part_img[xy].at<
463                                     cv::Vec3b>( part_img[0].rows-2, px_x)[
464                                         c])/2);
465
466                     }
467                     //各ピクセル左右
468                     for(int px_y=0; px_y < part_img[0].rows; ++px_y){
469                         // cost_tmp[DIRE_R] += abs(part_img[
470                             abs_xy].at<cv::Vec3b>( px_y, part_img
471                                 [0].cols-1)[c] - (part_img[abs_xy].at<
472                                     cv::Vec3b>( px_y, 0)[c] - part_img[xy
473                                         ].at<cv::Vec3b>( px_y, part_img[0].
474                                             cols-2)[c])/2);
475
476                         //
477                         abs(part_img[abs_xy].at<cv::Vec3b
478                             >( px_y, part_img[0].cols-1)[c] - (
479                             part_img[xy].at<cv::Vec3b>( px_y, 0)[c
480                                 ]);
481                         // cost_tmp[DIRE_L] += abs(part_img[
482                             abs_xy].at<cv::Vec3b>( px_y, 0)[c] -
483                             part_img[xy].at<cv::Vec3b>( px_y,
484                                 part_img[0].cols-1)[c]);
485                     }
486                 }
487             }
488         }
489     }

```

```

466                                     // 上下に対しては縦のピクセル数、左右に対しては横のピクセ
467                                     ル数
468                                     // を×る事によって、結合度の重みを長さに依存させない
469                                     // ワンちゃんオーバーフローが怖い(たぶん大丈夫)
470                                     cost_tmp[DIRE_U] *= part_img[0].rows;
471                                     cost_tmp[DIRE_D] *= part_img[0].rows;
472                                     cost_tmp[DIRE_R] *= part_img[0].cols;
473                                     cost_tmp[DIRE_L] *= part_img[0].cols;
474
475                                     // cost_t [コスト, 自分の座標, 相手の座標, 方向]
476                                     // cost [自分の座標][方向][相手の座標]
477                                     for(int k=0; k < 4; ++k){
478                                         cost_t_maru.push_back(make_tuple( cost_tmp[k],
479                                                                 abs_xy, xy, k));
480                                         cost_maru[abs_xy][k][xy] = make_pair( cost_tmp[k]
481                                                                 , xy);
482                                     }
483
484                                     cout << "start_sort" << endl;
485                                     // 独自のルールで COST_TUPLEをソート
486                                     sort( cost_t.begin(), cost_t.end(), my_compare);
487                                     cout << "end calc" << endl;
488     }

```

List 6: placement/PPMFILE.hpp

```

1  #include <vector>
2  #include <tuple>
3
4  #include <opencv2/core/core.hpp>
5
6  #include "../sort/PosData.h"
7  #define DEBUG
8
9  #define ORIGIN_IMG 0
10 #define LINE_IMG 1
11 #define RESULT_IMG 2
12
13 #define DIRE_U 0
14 #define DIRE_D 1
15 #define DIRE_R 2
16 #define DIRE_L 3
17
18
19 #define MIN_2( A, B ) ((A) < (B) ? (A) : (B))
20 #define BIG_2( A, B ) ((A) > (B) ? (A) : (B))
21
22 using namespace std;
23
24 typedef tuple< int, int, int, int> COST_TUPLE;
25
26 // スクラップ(いくつかのパーツの集まり)
27 typedef struct{
28     map<int,pair<int,int> > elements;           // パーツ
29     // idから、それが使われているか?
30     map<pair<int, int>, int> used_p; // ポジションから、そこに何があるか?
31 }SCRAP;
32
33 class PPMFILE{
34     private:
35         int part_size_x, part_size_y;          //左右どれだけのピース数があるか?
36

```

```

37         cv::Mat origin_img;
38         cv::Mat line_img;
39         cv::Mat result_img;
40
41         vector<cv::Mat> part_img;
42
43         // cost配列 (3次元)
44         vector< vector< vector< pair<int,int> > > > cost;
45         vector< vector< vector< pair<int,int> > > > cost_maru;
46         // cost_t配列 (1次元)
47         vector<COST_TUPLE> cost_t;
48         vector<COST_TUPLE> cost_t_maru;
49         // 配置の位置を示す配列
50         map<int,pair<int,int> > placement_pos;
51
52     public:
53
54         // 2次元を1次元に
55         inline int CONV_XY(int x,int y){
56             return x+y * part_size_x;
57         }
58         // XY座標を1次元に
59         inline int CONV_X(int XY){
60             return XY % part_size_x;
61         }
62         inline int CONV_Y(int XY){
63             return XY/part_size_x;
64         }
65         PPMFILE(cv::Mat origin_img_tmp, int piece_x, int piece_y);
66         //画像を表示する
67         void disp_img(int type);
68         //境界線を引く
69         void write_line(void);
70         // それぞれのピースに分割
71         void create_partition(void);
72         // 配列の近似値を計算 n する。
73         void calc_cost(void);
74         // まるさん方式
75         void calc_cost_maru(void);
76
77         void disp_cost_list(void);
78
79         // 配置
80         void placement(void);
81
82         void disp_placement(void);
83         //PosDataにデータを挿入
84         void set_PosData(PosData *data);
85
86         // 配置してみた時の表示
87         void create_result_img(void);
88
89         // xy_1のdire方向のxy_2とのcost
90         int get_cost(int xy_1, int xy_2, int dire);
91     };

```

List 7: placement/placement.cpp

```

1  #include <iostream>
2  #include <fstream>
3  #include <string>
4
5  #include <stdlib.h> /* abs 絶対値 */
6  #include <stdlib.h> /* rand() */
7  #include <time.h>   /* time */
8

```



```

9  #include <OpenGL/gl.h>
10 #include <GLUT/glut.h>
11
12 #define FILENAME "1.ppm"
13 #define PIECE_X 16
14 #define PIECE_Y 16
15
16 using namespace std;
17
18
19 void display(void);
20 void init(void);
21 void resize( int, int);
22 void Point( int, int, int, int, int, int);
23
24 class PPMFILE{
25     private:
26         int width, height, bright;
27
28         //image[col][x][y] (メモリの動的確保)
29         unsigned char ***image = new unsigned char **[3];
30     public:
31         void init_data(string filename){
32             // バイナリモードで開く
33             ifstream fin;
34             fin.open(filename, ios::in | ios::binary);
35
36             if(!fin){
37                 cout << "file : \"< filename << \"\"reading
                    failure.\" << endl;
38                 exit(1);
39             }else{
40                 cout << "file : \"< filename << \"\" reading
                    succes.\" << endl;
41             }
42
43             // 1行目の取得 P6:バイナリモード or P3:テキストモード
44             string line;
45             getline( fin, line);
46             cout << "format : " << line << endl;
47             // 3行目(width height の取得)
48             while (width == 0 || height == 0){
49                 string line;
50                 getline( fin,line);
51                 if(line.at(0) !='#')
52                     sscanf(line.c_str(),"%d %d",&width, &
                        height);
53             }
54             cout << "width: " << width << endl << "height: " <<
                height << endl;
55
56             //問題文形式だと、輝度で1行使っている
57             getline( fin, line);
58             sscanf(line.c_str(),"%d",&bright);
59             cout << "bright: " << bright << endl;
60
61             //動的画像保存領域の確保
62             for(int i=0; i<3; ++i){
63                 image[i] = new unsigned char*[width];
64                 for(int x=0; x<width; ++x){
65                     image[i][x] = new unsigned char[height];
66                 }
67             }
68
69             //画像の読み込み
70             for(int y=0; y<height; ++y){
71                 for(int x=0; x<width; ++x){

```

```

72         for(int col=0; col<3; ++col){
73             fin.read(reinterpret_cast<char
                    *>(&image[col][x][y]),1);
74         }
75     }
76 }
77
78     fin.close();
79 }
80
81 void delete_data(){
82     for(int i=0; i<3; ++i){
83         for(int x=0; x<width; ++x){
84             delete[] image[i][x];
85         }
86         delete[] image[i];
87     }
88     delete[] image;
89     cout << "create delete memory" << endl;
90 }
91
92 void disp_data(){
93     for(int y=0; y<height; ++y){
94         for(int x=0; x<width; ++x){
95             Point( x, y, 1, image[0][x][y],image[1][x
                    ][y],image[2][x][y]);
96             //赤だけ表示とか
97             //Point( x, y, 1, image[0][x][y],0,0);
98         }
99     }
100 }
101
102 //画面サイズをwidth,heightで初期化する
103 void imageGlutInitWindowSize(){
104     glutInitWindowSize( width, height);
105 }
106
107 int get_width(){
108     return width;
109 }
110 int get_height(){
111     return height;
112 }
113
114 void set_data(int x, int y, int col, unsigned char data){
115     image[col][x][y] = data;
116 }
117 int get_data(int x, int y, int col){
118     return image[col][x][y];
119 }
120 };
121
122 class PIECE_DATA{
123     private:
124         int piece_width, piece_height;
125         unsigned char ***piece = new unsigned char **[3];
126
127     public:
128         int rx, ry;
129         int lx, ly;
130         int ux, uy;
131         int dx, dy;
132         double rp, lp, up, dp;
133
134         int re_pos_x, re_pos_y;
135
136         void init_data(int width, int height);

```

```

137         void set_data(int x, int y, int col, unsigned char data);
138         int get_data(int x, int y, int col);
139         void delete_data();
140         void disp_data_pos(int , int);
141     };
142
143     void PIECE_DATA::init_data(int width, int height){
144         piece_width = width;
145         piece_height = height;
146
147         //動的画像保存領域の確保
148         for(int i=0; i<3; ++i){
149             piece[i] = new unsigned char*[piece_width];
150             for(int x=0; x<width; ++x){
151                 piece[i][x] = new unsigned char[piece_height];
152             }
153         }
154     }
155     void PIECE_DATA::set_data(int x, int y, int col, unsigned char data){
156         piece[col][x][y] = data;
157     }
158     int PIECE_DATA::get_data(int x, int y, int col){
159         return piece[col][x][y];
160     }
161     void PIECE_DATA::delete_data(){
162         for(int i=0; i<3; ++i){
163             for(int x=0; x<piece_width; ++x){
164                 delete[] piece[i][x];
165             }
166             delete[] piece[i];
167         }
168         delete[] piece;
169     }
170     void PIECE_DATA::disp_data_pos(int start_x, int start_y){
171         for (int y = 0; y < piece_height; y++) {
172             for (int x = 0; x < piece_width; x++) {
173                 Point( start_x+x, start_y+y, 1, piece[0][x][y], piece[1][
x][y], piece[2][x][y]);
174             }
175         }
176     }
177
178     PPMFILE pic_data;
179     PIECE_DATA **piece;
180
181     int piece_x;
182     int piece_y;
183
184     int piece_width, piece_height;
185
186     //OpenGL
187     void display(void)
188     {
189         glClear(GL_COLOR_BUFFER_BIT);
190         pic_data.disp_data();
191
192         for(int pos_y=0; pos_y<piece_y; pos_y++){
193             for(int pos_x=0; pos_x<piece_x; pos_x++){
194                 piece[pos_x][pos_y].disp_data_pos( piece[pos_x][pos_y].
re_pos_x*piece_width, piece[pos_x][pos_y].re_pos_y*
piece_height);
195                 cout << pos_x << ":" << pos_y << "|" << piece[pos_x][
pos_y].re_pos_x << ":" << piece[pos_x][pos_y].re_pos_y
<< endl;
196             }
197         }
198         glFlush();

```

```

199 }
200
201 //どれだけ隣り合うドット同士の色が近いのか
202 //数字分けはとりあえず適当
203 int compare_dot(int a, int b){
204     if (a==b)
205         return 10;
206     else if(abs(a-b) <= 5)
207         return 9;
208     else if(abs(a-b) <=10)
209         return 4;
210     else if(abs(a-b) <=30)
211         return 2;
212     else
213         return 0;
214 }
215
216
217 int main(int argc, char *argv[])
218 {
219     //乱数初期化
220     srand((unsigned)time(NULL));
221
222     pic_data.init_data(FILENAME);
223
224     //分割数が入る予定
225     piece_x = PIECE_X;
226     piece_y = PIECE_Y;
227
228     //画像の大きさ
229     int image_width = pic_data.get_width();
230     int image_height = pic_data.get_height();
231
232     //分割画像の大きさ
233     piece_width = image_width/piece_x;
234     piece_height = image_height/piece_y;
235
236     //各ピースの配列宣言
237     piece = new PIECE_DATA *[piece_x];
238     for(int i=0; i<piece_x; i++){
239         piece[i] = new PIECE_DATA[piece_y];
240     }
241
242     //各ピースのメモリの確保
243     for(int i=0; i<piece_x; i++){
244         for(int j=0; j<piece_y; j++){
245             piece[i][j].init_data( piece_width, piece_height);
246         }
247     }
248
249     //ピースごとに画像を取得
250     for(int def_y=0; def_y<piece_y; def_y++){
251         for(int def_x=0; def_x<piece_x; def_x++){
252
253             for(int y=0; y<piece_height; y++){
254                 for(int x=0; x<piece_width; x++){
255                     for(int col=0; col<3; col++){
256                         piece[def_x][def_y].set_data( x,
257                                                         y, col, pic_data.get_data( x+(
258                                                         piece_width*def_x), y+(
259                                                         piece_height*def_y), col));
260                     }
261                 }
262             }
263         }
264     }
265 }

```



```

310         compare_l[def_x][def_y] +=
            compare_dot( piece[pos_x][
311                pos_y].get_data( 0, y, 2),
312                piece[def_x][def_y].get_data(
313                    piece_width-1, y, 2));
314     }
    for(int x=0; x<piece_width; x++){
        //画像の上側
        compare_u[def_x][def_y] +=
            compare_dot( piece[pos_x][
315                pos_y].get_data( x, 0, 0),
            piece[def_x][def_y].get_data(
316                x, piece_height-1, 0));
        compare_u[def_x][def_y] +=
            compare_dot( piece[pos_x][
                pos_y].get_data( x, 0, 1),
            piece[def_x][def_y].get_data(
                x, piece_height-1, 1));
        compare_u[def_x][def_y] +=
            compare_dot( piece[pos_x][
                pos_y].get_data( x, 0, 2),
            piece[def_x][def_y].get_data(
                x, piece_height-1, 2));
        //画像の下側
        //cout << "chk_7" << endl;
        compare_d[def_x][def_y] +=
            compare_dot( piece[pos_x][
317                pos_y].get_data( x,
318                piece_height-1, 0), piece[
319                def_x][def_y].get_data( x, 0,
                    0));
        compare_d[def_x][def_y] +=
            compare_dot( piece[pos_x][
320                pos_y].get_data( x,
                piece_height-1, 1), piece[
                def_x][def_y].get_data( x, 0,
                    1));
        compare_d[def_x][def_y] +=
            compare_dot( piece[pos_x][
321                pos_y].get_data( x,
                piece_height-1, 2), piece[
                def_x][def_y].get_data( x, 0,
                    2));
    }
}

}

}

max_r = compare_r[0][0];
max_rx = max_ry = 0;
max_l = compare_l[0][0];
max_lx = max_ly = 0;
max_u = compare_u[0][0];
max_ux = max_uy = 0;
max_d = compare_d[0][0];
max_dx = max_dy = 0;

//それぞれのピースに対して
for(int def_y=0; def_y<piece_y; def_y++){
    for (int def_x = 0; def_x < piece_x; def_x++) {
        if(compare_r[def_x][def_y] > max_r){
            max_r = compare_r[def_x][def_y];
            max_rx = def_x;
            max_ry = def_y;
        }
        if(compare_l[def_x][def_y] > max_l){
            max_l = compare_l[def_x][def_y];

```

```

346         max_lx = def_x;
347         max_ly = def_y;
348     }
349     if(compare_u[def_x][def_y] > max_u){
350         max_u = compare_u[def_x][def_y];
351         max_ux = def_x;
352         max_uy = def_y;
353     }
354     if(compare_d[def_x][def_y] > max_d){
355         max_d = compare_d[def_x][def_y];
356         max_dx = def_x;
357         max_dy = def_y;
358     }
359 }
360 }
361
362 piece[pos_x][pos_y].rx = max_rx;
363 piece[pos_x][pos_y].ry = max_ry;
364 piece[pos_x][pos_y].rp = (max_r)/(double)(piece_height
    *3*10)*100;
365 //cout << max_r << ":aaa" << endl;
366
367 piece[pos_x][pos_y].lx = max_lx;
368 piece[pos_x][pos_y].ly = max_ly;
369 piece[pos_x][pos_y].lp = (max_l)/(double)(piece_height
    *3*10)*100;
370
371 piece[pos_x][pos_y].ux = max_ux;
372 piece[pos_x][pos_y].uy = max_uy;
373 piece[pos_x][pos_y].up = (max_u)/(double)(piece_width
    *3*10)*100;
374
375 piece[pos_x][pos_y].dx = max_dx;
376 piece[pos_x][pos_y].dy = max_dy;
377 piece[pos_x][pos_y].dp = (max_d)/(double)(piece_width
    *3*10)*100;
378
379 }
380 }
381
382 //結果表示
383 for(int pos_y=0; pos_y<piece_y; pos_y++){
384     for(int pos_x=0; pos_x<piece_x; pos_x++){
385         cout << "[" << pos_x << ":" << pos_y << "]" << endl;
386         cout << "R: (" << piece[pos_x][pos_y].rx << ":" << piece[
            pos_x][pos_y].ry << ")" << "[" << piece[pos_x][pos_y].
            rp << "]" << endl;
387         cout << "L: (" << piece[pos_x][pos_y].lx << ":" << piece[
            pos_x][pos_y].ly << ")" << "[" << piece[pos_x][pos_y].
            lp << "]" << endl;
388         cout << "U: (" << piece[pos_x][pos_y].ux << ":" << piece[
            pos_x][pos_y].uy << ")" << "[" << piece[pos_x][pos_y].
            up << "]" << endl;
389         cout << "D: (" << piece[pos_x][pos_y].dx << ":" << piece[
            pos_x][pos_y].dy << ")" << "[" << piece[pos_x][pos_y].
            dp << "]" << endl;
390     }
391 }
392
393 //ここから並び替え
394 int datum_x, datum_y;
395
396 int r1p, r2p;
397 int r1x,r1y, r2x,r2y;
398
399 do{
400

```

```

401 //適当に基準となるピースを選択
402 datum_x = rand()%piece_x;
403 datum_y = rand()%piece_y;
404
405 cout << "dx: " << datum_x << " dy: " << datum_y << endl;
406
407 //自分の一つ右のピースの
408 r1p = piece[datum_x][datum_y].rp;
409 r1x = piece[piece[datum_x][datum_y].rx][piece[datum_x][datum_y].
    ry].dx;
410 r1y = piece[piece[datum_x][datum_y].rx][piece[datum_x][datum_y].
    ry].dy;
411
412 r2p = piece[datum_x][datum_y].dp;
413 r2x = piece[piece[datum_x][datum_y].dx][piece[datum_x][datum_y].
    dy].rx;
414 r2y = piece[piece[datum_x][datum_y].dx][piece[datum_x][datum_y].
    dy].ry;
415 }while(r1x != r2x || r1y != r2y || r1p <= 50 || r2p <= 50);
416
417 //まずは、左に行けるところまで
418 do{
419     cout << "dx: " << datum_x << " dy: " << datum_y << endl;
420
421     r1p = piece[datum_x][datum_y].lp;
422     r1x = piece[piece[datum_x][datum_y].lx][piece[datum_x][datum_y].
        ly].dx;
423     r1y = piece[piece[datum_x][datum_y].lx][piece[datum_x][datum_y].
        ly].dy;
424
425     r2p = piece[piece[datum_x][datum_y].dx][piece[datum_x][datum_y].
        dy].lp;
426     r2x = piece[piece[datum_x][datum_y].dx][piece[datum_x][datum_y].
        dy].lx;
427     r2y = piece[piece[datum_x][datum_y].dx][piece[datum_x][datum_y].
        dy].ly;
428
429     if((r1x == r2x) && (r1y == r2y) && (r1p >= 50 || r2p >= 50)){
430         cout << "left" << endl;
431         int tmp;
432         tmp = datum_x;
433         datum_x = piece[datum_x][datum_y].lx;
434         datum_y = piece[tmp][datum_y].ly;
435     }else{
436         break;
437     }
438 }while(1);
439
440 //次に、上に行ける所まで
441 do{
442     cout << "dx: " << datum_x << " dy: " << datum_y << endl;
443
444     r1p = piece[datum_x][datum_y].up;
445     r1x = piece[piece[datum_x][datum_y].ux][piece[datum_x][datum_y].
        uy].rx;
446     r1y = piece[piece[datum_x][datum_y].ux][piece[datum_x][datum_y].
        uy].ry;
447
448     r2p = piece[piece[datum_x][datum_y].rx][piece[datum_x][datum_y].
        ry].up;
449     r2x = piece[piece[datum_x][datum_y].rx][piece[datum_x][datum_y].
        ry].ux;
450     r2y = piece[piece[datum_x][datum_y].rx][piece[datum_x][datum_y].
        ry].uy;
451
452     if((r1x == r2x) && (r1y == r2y) && (r1p >= 50 || r2p >= 50)){
453         cout << "up" << endl;

```



```

454             int tmp;
455             tmp = datum_x;
456             datum_x = piece[datum_x][datum_y].ux;
457             datum_y = piece[tmp][datum_y].uy;
458         }else{
459             break;
460         }
461     }while(1);
462
463     //左上に来たため、順番に右に進んでいく
464     int xx, yy;
465     xx = datum_x;
466     yy = datum_y;
467
468     for(int pos_y=0; pos_y<piece_y; pos_y++){
469         for(int pos_x=0; pos_x<piece_x; pos_x++){
470             cout << xx << "," << yy << "=" << pos_x << "," << pos_y
471                 << endl;
472             piece[xx][yy].re_pos_x = pos_x;
473             piece[xx][yy].re_pos_y = pos_y;
474
475             int tmp;
476             tmp = xx;
477             xx = piece[xx][yy].rx;
478             yy = piece[tmp][yy].ry;
479         }
480         xx = piece[datum_x][datum_y].dx;
481         yy = piece[datum_x][datum_y].dy;
482         datum_x = xx;
483         datum_y = yy;
484     }
485     //glutInitWindowPosition(100, 100);
486
487     pic_data.imageGlutInitWindowSize();
488     glutInit(&argc, argv);
489     glutInitDisplayMode(GLUT_RGBA);
490     glutCreateWindow("create_image");
491     glutReshapeFunc(resize);
492     init();
493
494     glutDisplayFunc(display);
495
496     glutMainLoop();
497
498     //メモリの解放呼び出されて無いかも。。。
499     pic_data.delete_data();
500     for(int i=0; i<piece_y; i++){
501         for(int j=0; j<piece_x; j++){
502             piece[j][i].delete_data();
503         }
504     }
505     for(int i=0; i<piece_x; i++){
506         delete[] piece[i];
507     }
508     delete[] piece;
509     cout << "create delete PIECE memory" << endl;
510
511     for(int i=0; i<piece_x; i++){
512         delete[] compare_r[i];
513         delete[] compare_l[i];
514         delete[] compare_u[i];
515         delete[] compare_d[i];
516     }
517     delete[] compare_r;
518     delete[] compare_l;
519     delete[] compare_u;
520     delete[] compare_d;

```

```

520         cout << "create delete compare memory" << endl;
521     }

```

List 8: sort/Pos.cpp

```

1  #include "Pos.h"
2  // ポジション
3
4  Pos::Pos() {
5      x = y = -1;
6  }
7
8  Pos::Pos(int x, int y) {
9      this->x = x;
10     this->y = y;
11 }
12
13 void Pos::setZero() {
14     this->x = this->y = 0;
15 }

```

List 9: sort/Pos.h

```

1  #ifndef INCLUDED_POS_H
2  #define INCLUDED_POS_H
3
4  class Pos {
5  public:
6      int x, y;
7
8      Pos();
9      Pos(int x, int y);
10     void setZero();
11 };
12
13 #endif

```

List 10: sort/PosData.cpp

```

1  #include "PosData.h"
2  #include "stdio.h"
3  #include "stdlib.h"
4  #include "util.h"
5
6  PosData::PosData() {
7  }
8
9  PosData::PosData(int w, int h) {
10     int i;
11
12     width = w;
13     height = h;
14     data = new Pos*[height];
15     for(i = 0; i < height; i++) {
16         data[i] = new Pos[width];
17     }
18 }
19
20 PosData::~PosData() {
21     int i;
22
23     for(i = 0; i < height; i++) {
24         delete [] data[i];
25     }

```

```

26     delete [] data;
27 }
28
29 void PosData::dispData() {
30     int i, j;
31
32     for(i = 0; i < height; i++) {
33         for(j = 0; j < width; j++) {
34             printf("%X%X ", data[i][j].x, data[i][j].y);
35         }
36         puts("");
37     }
38     puts("");
39 }
40
41 int PosData::getHeight() {
42     return this->height;
43 }
44 int PosData::getWidth() {
45     return this->width;
46 }
47
48 int PosData::getX(int ox, int oy) {
49     if(!checkInScope(width, height, ox, oy)) myerror(1);
50     return data[oy][ox].x;
51 }
52
53 int PosData::getY(int ox, int oy) {
54     if(!checkInScope(width, height, ox, oy)) myerror(1);
55     return data[oy][ox].y;
56 }
57
58 void PosData::randomizeData() {
59     int i, j, x, y;
60
61     for(i = 0; i < height; i++) {
62         for(j = 0; j < width; j++) {
63             x = rand() % width;
64             y = rand() % height;
65             swapPos(&data[i][j], &data[y][x]);
66         }
67     }
68 }
69
70 void PosData::setX(int ox, int oy, int x) {
71     if(!checkInScope(width, height, ox, oy)) myerror(1);
72     data[oy][ox].x = x;
73 }
74
75 void PosData::setY(int ox, int oy, int y) {
76     if(!checkInScope(width, height, ox, oy)) myerror(1);
77     data[oy][ox].y = y;
78 }
79
80 void PosData::setData(int ox, int oy, int x, int y) {
81     data[oy][ox].x = x;
82     data[oy][ox].y = y;
83 }

```

List 11: sort/PosData.h

```

1  #ifndef INCLUDED_POSDATA_H
2  #define INCLUDED_POSDATA_H
3  #include "Pos.h"
4
5  class PosData {

```

```

6 private:
7     Pos **data;
8     int width, height;
9
10 public:
11     PosData();
12     PosData(int w, int h);
13     ~PosData();
14     void dispData();
15     int getHeight();
16     int getWidth();
17     int getX(int ox, int oy);
18     int getY(int ox, int oy);
19     void randomizeData();
20     void setX(int ox, int oy, int x);
21     void setY(int ox, int oy, int y);
22     void setData(int ox, int oy, int x, int y);
23 };
24
25 #endif

```

List 12: sort/Process5.cpp

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "Process5.h"
4  #include "util.h"
5
6  using namespace std;
7
8  Pro5::Process5(): ProcessBase() {
9      target.setZero();
10     target_data.setZero();
11 }
12
13 Pro5::Process5(int w, int h): ProcessBase(w, h) {
14     target.setZero();
15     target_data.setZero();
16 }
17
18 void Pro5::dispSorted() {
19     list<Pos>::iterator p;
20
21     for(p = sorted.begin(); p != sorted.end(); p++) {
22         printf("%d, %d\n", p->x, p->y);
23     }
24     puts("");
25 }
26
27 void Pro5::dispSortedData() {
28     list<Pos>::iterator p;
29
30     int i, j;
31
32     for(i = 0; i < table->getHeight(); i++) {
33         for(j = 0; j < table->getWidth(); j++) {
34             for(p = sorted.begin(); p != sorted.end(); p++) {
35                 if(checkPosEqual(j, i, p->x, p->y))
36                     changeWordColor(RED);
37             }
38             if(checkPosEqual(j, i, table->getSelected().x, table->getSelected().y))
39                 changeWordColor(GREEN);
40             printf("%X%X ", table->getData(j, i).x, table->getData(j, i).y);
41             defaultWordColor();
42         }
43         puts("");

```

```

44     }
45     puts("");
46 }
47
48 // 安全
49 int Pro5::isSelectedNextToTarget() {
50     Pos s = table->getSelected();
51
52     return isNext(s, target);
53 }
54
55 int Pro5::isSorted(int y) {
56     int x;
57     Pos data;
58
59     for(x = 0; x < table->getWidth(); x++) {
60         data = table->getData(x, y);
61         if(data.x != x || data.y != y) return 0;
62     }
63     return 1;
64 }
65
66 void Pro5::moveSelected(Pos destination) {
67     Pos s = table->getSelected();
68     list<Pos>::iterator p;
69
70     int directionLR;
71     int directionUD;
72     int move_dir = 0;
73     int move_flag = 0;
74     int half;
75     int old_direction = -1;
76
77     puts("-----moveSelected-----");
78
79     while(!checkPosEqual(destination, s)) {
80         s = table->getSelected();
81         //half = s.y / (table->getHeight() / 2);
82         half = 1;
83         directionLR = getDirectionLR(s.x, destination.x);
84         directionUD = getDirectionUD(s.y, destination.y);
85         table->dispData();
86         if(half == 1) {
87             if(directionLR == EQUAL) {
88                 move_dir = directionUD;
89                 move_flag = UD;
90             } else {
91                 move_dir = directionLR;
92                 move_flag = LR;
93             }
94         } else if(half == 0) {
95             if(directionUD == EQUAL) {
96                 move_dir = directionLR;
97                 move_flag = LR;
98             } else {
99                 move_dir = directionUD;
100                 move_flag = UD;
101             }
102         }
103         for(p = sorted.begin(); p != sorted.end(); p++) {
104             if(checkPosEqual(surroundings(s, move_dir), *p)) {
105                 if(move_flag == LR) {
106                     move_dir = directionUD;
107                     move_flag = UD;
108                     break;
109                 } else if(move_flag == UD) {
110                     move_dir = directionLR;

```

```

111         move_flag = LR;
112         break;
113     }
114 }
115 }
116 for(p = sorted.begin(); p != sorted.end(); p++) {
117     if(checkPosEqual(surroundings(s, move_dir), *p)) {
118         if(move_flag == LR) {
119             move_dir = getReversedDirection(directionLR);
120             break;
121         } else if(move_flag == UD) {
122             move_dir = getReversedDirection(directionUD);
123             break;
124         }
125     }
126 }
127 for(p = sorted.begin(); p != sorted.end(); p++) {
128     if(checkPosEqual(surroundings(s, move_dir), *p)) {
129         if(move_flag == LR) {
130             move_dir = getReversedDirection(directionUD);
131             move_flag = UD;
132             break;
133         } else if(move_flag == UD) {
134             move_dir = getReversedDirection(directionLR);
135             move_flag = LR;
136             break;
137         }
138     }
139 }
140 /*
141  if(old_direction == move_dir) {
142      move_dir = getReversedDirection(move_dir);
143  }*/
144 printf("move_dir = %d\n", move_dir);
145 table->swapSelected(move_dir);
146 table->dispData(target.x, target.y);
147 old_direction = getReversedDirection(move_dir);
148 }
149 puts("=====moveSelected end=====");
150 }
151
152 void Pro5::moveSelectedNextTarget() {
153     Pos s = table->getSelected();
154     list<Pos>::iterator p;
155
156     int directionLR;
157     int directionUD;
158     int move_dir = 0;
159     int move_flag = 0;
160     int half = 1;
161     int old_direction = -1;
162
163     puts("-----moveSelectedNextTarget-----");
164     table->dispData(target.x, target.y);
165
166     while(!isSelectedNextToTarget()) {
167         s = table->getSelected();
168         directionLR = getDirectionLR(s.x, target.x);
169         directionUD = getDirectionUD(s.y, target.y);
170         if(half == 1) {
171             if(isNextX(target, s)) {
172                 move_dir = directionUD;
173                 move_flag = UD;
174             } else {
175                 move_dir = directionLR;
176                 move_flag = LR;
177             }

```

```

178     } else if(half == 0) {
179         if(isNextY(target, s)) {
180             move_dir = directionLR;
181             move_flag = LR;
182         } else {
183             move_dir = directionUD;
184             move_flag = UD;
185         }
186     }
187     for(p = sorted.begin(); p != sorted.end(); p++) {
188         if(checkPosEqual(surroundings(s, move_dir), *p)) {
189             if(move_flag == LR) {
190                 move_dir = directionUD;
191                 move_flag = UD;
192                 break;
193             } else if(move_flag == UD) {
194                 move_dir = directionLR;
195                 move_flag = LR;
196                 break;
197             }
198         }
199     }
200     for(p = sorted.begin(); p != sorted.end(); p++) {
201         if(checkPosEqual(surroundings(s, move_dir), *p)) {
202             if(move_flag == LR) {
203                 move_dir = getReversedDirection(directionLR);
204                 break;
205             } else if(move_flag == UD) {
206                 move_dir = getReversedDirection(directionUD);
207                 break;
208             }
209         }
210     }
211     for(p = sorted.begin(); p != sorted.end(); p++) {
212         if(checkPosEqual(surroundings(s, move_dir), *p)) {
213             if(move_flag == LR) {
214                 move_dir = getReversedDirection(directionUD);
215                 move_flag = UD;
216                 break;
217             } else if(move_flag == UD) {
218                 move_dir = getReversedDirection(directionLR);
219                 move_flag = LR;
220                 break;
221             }
222         }
223     }
224     /*
225     for(p = sorted.begin(); p != sorted.end(); p++) {
226         if(checkPosEqual(surroundings(s, move_dir), *p)) {
227             if(move_flag == LR) {
228                 move_dir = directionUD;
229                 move_flag = UD;
230                 break;
231             } else if(move_flag == UD) {
232                 move_dir = directionLR;
233                 move_flag = LR;
234                 break;
235             }
236         }
237     }
238     if(old_direction == move_dir) {
239         move_dir = getReversedDirection(move_dir);
240     }
241     for(p = sorted.begin(); p != sorted.end(); p++) {
242         if(checkPosEqual(surroundings(s, move_dir), *p)) {
243             if(move_flag == LR) {
244                 move_dir = getReversedDirection(directionUD);

```

```

245         move_flag = UD;
246         break;
247     } else if(move_flag == UD) {
248         move_dir = getReversedDirection(directionLR);
249         move_flag = LR;
250         break;
251     }
252 }
253 }*/
254 table->swapSelected(move_dir);
255 table->dispData(target.x, target.y);
256 old_direction = getReversedDirection(move_dir);
257 }
258 puts("=====moveSelectedNextTarget=====");
259 }
260
261 int Pro5::moveTarget(Pos pos) {
262     list<Pos>::iterator p;
263
264     int directionLR;
265     int directionUD;
266     int move_dir = 0;
267     int move_flag = 0;
268     //int half = target.y / (table->getHeight() / 2);
269     int old_direction = -1;
270     int old_flag = -1;
271
272     puts("-----moveTarget-----");
273
274     while(!checkPosEqual(target, pos)) {
275         target = table->findData(target_data);
276         directionLR = getDirectionLR(target.x, pos.x);
277         directionUD = getDirectionUD(target.y, pos.y);
278         table->dispData();
279         if(old_flag == LR) {
280             if(directionUD == EQUAL) {
281                 move_dir = directionLR;
282                 move_flag = LR;
283             } else {
284                 move_dir = directionUD;
285                 move_flag = UD;
286             }
287         } else {
288             if(directionLR == EQUAL) {
289                 move_dir = directionUD;
290                 move_flag = UD;
291             } else {
292                 move_dir = directionLR;
293                 move_flag = LR;
294             }
295         }
296         for(p = sorted.begin(); p != sorted.end(); p++) {
297             if(checkPosEqual(surroundings(target, move_dir), *p)) {
298                 if(move_flag == LR) {
299                     move_dir = directionUD;
300                     move_flag = UD;
301                     break;
302                 } else if(move_flag == UD) {
303                     move_dir = directionLR;
304                     move_flag = LR;
305                     break;
306                 }
307             }
308         }
309         if(old_direction == move_dir) {
310             move_dir = getReversedDirection(move_dir);
311         }

```



```

312     for(p = sorted.begin(); p != sorted.end(); p++) {
313         if(checkPosEqual(surroundings(target, move_dir), *p)) {
314             if(move_flag == LR) {
315                 move_dir = getReversedDirection(directionUD);
316                 move_flag = UD;
317                 break;
318             } else if(move_flag == UD) {
319                 move_dir = getReversedDirection(directionLR);
320                 move_flag = LR;
321                 break;
322             }
323         }
324     }
325     printf("directionaaaaa%d, %d, %d\n", directionLR, directionUD, move_dir);
326     moveSelectedNextTarget();
327     rotateSelected(move_dir);
328     table->swapSelected(getReversedDirection(move_dir));
329     table->dispData(target.x, target.y);
330     old_direction = getReversedDirection(move_dir);
331     old_flag = move_flag;
332 }
333 puts("====moveTarget end====");
334 return 0;
335 }
336
337 void Pro5::rotateSelected(int direction) {
338     // directionは動かす方向
339     // targetから見たselectedの方向
340     int dir_selected = getDirection(target, table->getSelected());
341     int move_distance = abs(direction - dir_selected);
342     // 1か-1
343     // 間違ってる
344     //-----直すところ
345     int move_direction = (dir_selected > direction) ? -1 : 1;
346
347     puts("-----rotateSelected-----");
348     printf("direction = %d\n", direction);
349     printf("target = %d, %d\n", target.x, target.y);
350     if(direction == EQUAL) return;
351     if(checkPosEqual(table->getSelected(), surroundings(target, direction))) {
352         puts("selected dont need to rotate");
353         return;
354     }
355     if(!checkInScope(table->getWidth(), table->getHeight(), surroundings(target,
356         direction).x, surroundings(target, direction).y)) {
357         puts("cant rotate(move target)...");
358         return;
359     }
360     if(move_distance > DIRECTION_NUM / 2) {
361         puts("rotate distance is too long");
362         move_distance = DIRECTION_NUM - move_distance;
363         move_direction *= -1;
364     }
365
366     Pos dummy_target = target;
367     int i;
368     int reverse_flag = 0;
369     list<Pos>::iterator p;
370
371     for(i = dir_selected; i != direction; i = (i + move_direction + DIRECTION_NUM)
372         % DIRECTION_NUM) {
373         if(!checkInScope(table->getWidth(), table->getHeight(), surroundings(
374             dummy_target, i).x, surroundings(dummy_target, i).y)) reverse_flag = 1;
375         for(p = sorted.begin(); p != sorted.end(); p++) {
376             if(checkPosEqual(surroundings(dummy_target, i), *p)) {
377                 reverse_flag = 1;

```

```

376         break;
377     }
378 }
379 if(reverse_flag) {
380     move_distance = DIRECTION_NUM - move_distance;
381     move_direction *= -1;
382     break;
383 }
384 }
385
386 int move_dir;
387 int j = dir_selected;
388 printf("move_direction = %d\n", move_direction);
389 printf("move_distance = %d\n", move_distance);
390 for(i = 0; i < move_distance; i++) {
391     // 目的地についたら止めたほうがいいかも
392     j = (j + move_direction + DIRECTION_NUM) % DIRECTION_NUM;
393     move_dir = getDirection(table->getSelected(), surroundings(target, j));
394     printf("j = %d\n", j);
395     printf("move_dir = %d\n", move_dir);
396     table->swapSelected(move_dir);
397     table->dispData(target.x, target.y);
398 }
399 puts("====rotate end====");
400 }
401
402 string Pro5::sort() {
403     table->dispData();
404     // 一番右下になるデータを選択
405     table->findAndSelectData(table->getWidth()-1, table->getHeight()-1);
406     table->dispData();
407
408     sortUp();
409     dispSortedData();
410     sortDown();
411     dispSortedData();
412     table->dispData();
413     table->dispCost();
414
415     return table->getStringSortData();
416 }
417
418 void Pro5::sortDown() {
419     int i;
420
421     if(isSorted(table->getHeight()-2) && isSorted(table->getHeight()-1)) return;
422     for(i = 0; i < table->getWidth()-2; i++) {
423         Pos dummy = table->findData(Pos(i, table->getHeight()-2));
424         if(dummy.x >= i && dummy.x < i+2) {
425             target_data = Pos(i, table->getHeight()-2);
426             target = table->findData(target_data);
427             moveTarget(Pos(i+2, dummy.y));
428         }
429         target_data = Pos(i, table->getHeight()-1);
430         target = table->findData(target_data);
431         moveTarget(Pos(i, table->getHeight()-2));
432         sorted.push_back(Pos(i, table->getHeight()-2));
433         target_data = Pos(i, table->getHeight()-2);
434         target = table->findData(target_data);
435         moveTarget(Pos(i+1, table->getHeight()-2));
436         sorted.push_back(Pos(i+1, table->getHeight()-2));
437
438         moveSelected(Pos(i, table->getHeight()-1));
439         table->swapSelected(UP);
440         table->swapSelected(RIGHT);
441         sorted.pop_back();
442         sorted.pop_back();

```

```

443     sorted.push_back(Pos(i, table->getHeight()-1));
444     sorted.push_back(Pos(i, table->getHeight()-2));
445     table->dispData();
446 }
447 target_data = Pos(table->getWidth()-2, table->getHeight()-2);
448 target = table->findData(target_data);
449 moveTarget(target_data);
450 sorted.push_back(Pos(table->getWidth()-2, table->getHeight()-2));
451 table->dispData();
452
453 target_data = Pos(table->getWidth()-2, table->getHeight()-1);
454 target = table->findData(target_data);
455 table->dispData(target.x, target.y);
456 if(checkPosEqual(target, Pos(table->getWidth()-1, table->getHeight()-2))) {
457     table->selectData(target.x, target.y);
458     moveSelected(target_data);
459 }
460 target_data = Pos(table->getWidth()-1, table->getHeight()-2);
461 target = table->findData(target_data);
462 table->dispData(target.x, target.y);
463 if(checkPosEqual(target, Pos(table->getWidth()-2, table->getHeight()-1))) {
464     table->selectData(target.x, target.y);
465     moveSelected(target_data);
466 }
467 target_data = Pos(table->getWidth()-1, table->getHeight()-1);
468 target = table->findData(target_data);
469 table->selectData(target.x, target.y);
470 moveSelected(target_data);
471 }
472
473 void Pro5::sortUp() {
474     int i, j;
475     // 上半分
476     for(i = 0; i < table->getHeight()-2; i++) {
477         // 端以外の
478         if(isSorted(i)) {
479             puts("continue1");
480             for(j = 0; j < table->getWidth(); j++) {
481                 sorted.push_back(Pos(j, i));
482             }
483             continue;
484         }
485         for(j = 0; j < table->getWidth()-2; j++) {
486             target_data = Pos(j, i);
487             target = table->findData(target_data);
488             table->dispData(target.x, target.y);
489             // 目的地へ移動
490             moveTarget(target_data);
491             target = table->findData(target_data);
492             sorted.push_back(target);
493         }
494         // もし、1列全部揃ってたら continue
495         if(isSorted(i)) {
496             puts("continue2");
497             sorted.push_back(Pos(table->getWidth()-2, i));
498             sorted.push_back(Pos(table->getWidth()-1, i));
499             continue;
500         }
501         Pos dummy = table->findData(Pos(table->getWidth()-2, i));
502         if(dummy.x >= table->getWidth()-2 && dummy.x < table->getWidth() && dummy.y
           >= i && dummy.y < i+2) {
503             target_data = Pos(table->getWidth()-2, i);
504             target = table->findData(target_data);
505             moveTarget(Pos(table->getWidth()-2, i+2));
506         }
507         // もし、width-2のところに20が来てしまった時は20を下に移す
508         if(!checkPosEqual(table->getData(table->getWidth()-2, i), Pos(table->getWidth

```

```

509         (-1, i)) || !checkPosEqual(table->getData(table->getWidth()-2, i+1), Pos(
510         table->getWidth()-2, i))) {
511         target_data = Pos(table->getWidth()-1, i);
512         target = table->findData(target_data);
513         moveTarget(Pos(table->getWidth()-2, i));
514         sorted.push_back(Pos(table->getWidth()-2, i));
515         target_data = Pos(table->getWidth()-2, i);
516         target = table->findData(target_data);
517         moveTarget(Pos(table->getWidth()-2, i+1));
518         sorted.push_back(Pos(table->getWidth()-2, i+1));
519     } else {
520     }
521     moveSelected(Pos(table->getWidth()-1, i));
522     table->swapSelected(LEFT);
523     table->swapSelected(DOWN);
524     sorted.pop_back();
525     sorted.pop_back();
526     sorted.push_back(Pos(table->getWidth()-2, i));
527     sorted.push_back(Pos(table->getWidth()-1, i));
528     table->dispData();
529 }
530 }

```

List 13: sort/Process5.h

```

1  #ifndef INCLUDED_PROCESS5_H
2  #define INCLUDED_PROCESS5_H
3  #define LR 0
4  #define UD 1
5  #include "ProcessBase.h"
6  #include <list>
7
8  using namespace std;
9
10 typedef class Process5 : public ProcessBase {
11 private:
12     Pos target;
13     Pos target_data;
14     list<Pos> sorted;
15
16 private:
17     Process5();
18     void dispSorted();
19     void dispSortedData();
20     int isSelectedNextToTarget();
21     int isSorted(int y);
22     void moveSelected(Pos destination); // なにか間違っているかも
23     void moveSelectedNextTarget();
24     int moveTarget(Pos pos);
25     void rotateSelected(int direction);
26     void sortDown();
27     void sortUp();
28 public:
29     Process5(int w, int h);
30     string sort();
31 } Pro5;
32
33 // 指定した座標に, sortedを使いながらselectedを移動するメソッドを追加する.
34
35 #endif

```

List 14: sort/ProcessBase.cpp

```

1  #include <stdio.h>
2  #include "ProcessBase.h"

```

```

3
4 ProBase::ProcessBase() {
5 }
6
7 ProBase::ProcessBase(int width, int height) {
8     table = new Dataset(width, height);
9     table->randomizeData();
10 }
11
12 ProBase::~ProcessBase() {
13     delete table;
14 }
15
16 void ProBase::importData(PosData &data) {
17     table->importData(data);
18 }

```

List 15: sort/ProcessBase.h

```

1 #ifndef INCLUDED_PROCESSBASE_H
2 #define INCLUDED_PROCESSBASE_H
3 #include "dataset.h"
4
5 typedef class ProcessBase {
6 protected:
7     Dataset *table;
8
9 protected:
10     ProcessBase();
11 public:
12     ProcessBase(int width, int height);
13     ~ProcessBase();
14     void importData(PosData &data);
15 } ProBase;
16
17 #endif

```

List 16: sort/util.cpp

```

1 #include <stdio.h>
2 #include <iostream>
3 #include <string>
4 #include <stdlib.h>
5 #include "util.h"
6
7 using namespace std;
8
9 int checkPosEqual(int x1, int y1, int x2, int y2) {
10     return ((x1 == x2) && (y1 == y2));
11 }
12
13 int checkPosEqual(Pos p1, Pos p2) {
14     return checkPosEqual(p1.x, p1.y, p2.x, p2.y);
15 }
16
17 int getDirection(Pos source, Pos destination) {
18     if(source.x == destination.x && source.y == destination.y) {
19         return EQUAL;
20     } else if(source.x == destination.x && source.y > destination.y) {
21         return UP;
22     } else if(source.x < destination.x && source.y > destination.y) {
23         return UPPERRIGHT;
24     } else if(source.x < destination.x && source.y == destination.y) {
25         return RIGHT;
26     } else if(source.x < destination.x && source.y < destination.y) {

```

```

27     return LOWERRIGHT;
28 } else if(source.x == destination.x && source.y < destination.y) {
29     return DOWN;
30 } else if(source.x > destination.x && source.y < destination.y) {
31     return LOWERLEFT;
32 } else if(source.x > destination.x && source.y == destination.y) {
33     return LEFT;
34 } else if(source.x > destination.x && source.y > destination.y) {
35     return UPPERLEFT;
36 }
37     return EQUAL;
38 }
39
40 int getReversedDirection(int direction) {
41     return (direction + DIRECTION_NUM / 2) % DIRECTION_NUM;
42 }
43
44 int getDirectionLR(int ox, int x) {
45     if(ox == x) return EQUAL;
46     return (ox < x) ? RIGHT : LEFT;
47 }
48
49 int getDirectionUD(int oy, int y) {
50     if(oy == y) return EQUAL;
51     return (oy < y) ? DOWN : UP;
52 }
53
54 string getDirectionChar(int direction) {
55     switch(direction) {
56     case UP:
57         return "U";
58         break;
59     case RIGHT:
60         return "R";
61         break;
62     case DOWN:
63         return "D";
64         break;
65     case LEFT:
66         return "L";
67         break;
68     }
69     return "";
70 }
71
72 int isConnected(Pos p1, Pos p2) {
73     return ((p1.x-1 == p2.x && p1.y == p2.y) || (p1.x+1 == p2.x && p1.y == p2.y) ||
74             (p1.x == p2.x && p1.y-1 == p2.y) || (p1.x == p2.x && p1.y+1 == p2.y));
75 }
76
77 int isNext(Pos p1, Pos p2) {
78     return ((p1.x+1 == p2.x || p1.x-1 == p2.x || p1.x == p2.x) && (p1.y-1 == p2.y
79         || p1.y+1 == p2.y || p1.y == p2.y));
80 }
81
82 int isNextX(Pos p1, Pos p2) {
83     return (p1.x+1 == p2.x || p1.x-1 == p2.x || p1.x == p2.x);
84 }
85
86 int isNextY(Pos p1, Pos p2) {
87     return (p1.y-1 == p2.y || p1.y+1 == p2.y || p1.y == p2.y);
88 }
89
90 void myerror(int error_code) {
91     puts("MYERROR");
92     exit(1);
93 }

```

```

92
93 // directionの方向にxとyを移動する
94 void surroundings(int *x, int *y, int direction) {
95     switch(direction) {
96         case UP:
97             (*y)--;
98             break;
99         case RIGHT:
100             (*x)++;
101             break;
102         case DOWN:
103             (*y)++;
104             break;
105         case LEFT:
106             (*x)--;
107             break;
108         case UPPERRIGHT:
109             (*x)++;
110             (*y)--;
111             break;
112         case LOWERRIGHT:
113             (*x)++;
114             (*y)++;
115             break;
116         case LOWERLEFT:
117             (*x)--;
118             (*y)++;
119             break;
120         case UPPERLEFT:
121             (*x)--;
122             (*y)--;
123             break;
124     }
125 }
126
127 Pos surroundings(Pos Data, int direction) {
128     switch(direction) {
129         case UP:
130             Data.y--;
131             break;
132         case RIGHT:
133             Data.x++;
134             break;
135         case DOWN:
136             Data.y++;
137             break;
138         case LEFT:
139             Data.x--;
140             break;
141         case UPPERRIGHT:
142             Data.x++;
143             Data.y--;
144             break;
145         case LOWERRIGHT:
146             Data.x++;
147             Data.y++;
148             break;
149         case LOWERLEFT:
150             Data.x--;
151             Data.y++;
152             break;
153         case UPPERLEFT:
154             Data.x--;
155             Data.y--;
156             break;
157     }
158     return Data;

```

```

159 }
160
161 // num1とnum2の値を交換
162 void swapNum(int *num1, int *num2) {
163     int dummy = *num1;
164     *num1 = *num2;
165     *num2 = dummy;
166 }
167
168 // p1とp2の値を交換する
169 void swapPos(Pos *p1, Pos *p2) {
170     Pos dummy = *p1;
171     *p1 = *p2;
172     *p2 = dummy;
173 }

```

List 17: sort/util.h

```

1  #ifndef INCLUDED_UTIL_H
2  #define INCLUDED_UTIL_H
3  #include "Pos.h"
4  #include <stdio.h>
5  #include <iostream>
6  #include <string>
7
8  using namespace std;
9
10 #define EQUAL -1
11 #define UP 0
12 #define UPPERRIGHT 1
13 #define RIGHT 2
14 #define LOWERRIGHT 3
15 #define DOWN 4
16 #define LOWERLEFT 5
17 #define LEFT 6
18 #define UPPERLEFT 7
19 #define DIRECTION_NUM 8
20 #define BLACK 0
21 #define RED 1
22 #define GREEN 2
23 #define YELLOW 3
24 #define BLUE 4
25 #define MAGENTA 5
26 #define CYAN 6
27 #define WHITE 7
28
29 #define changeWordColor(cc) printf("\033[3%dm", cc)
30 #define changeBackColor(cc) printf("\033[4%dm", cc)
31 #define checkInScope(width, height, x, y) (x>=0 && x<width && y >=0 && y<height)
32 #define convertHex(x, y) x+y*0x10
33 #define convertX(num) num%0x10
34 #define convertY(num) num/0x10
35 #define defaultWordColor() printf("\033[39m")
36 #define defaultBackColor() printf("\033[49m")
37
38 extern int checkPosEqual(int x1, int y1, int x2, int y2);
39 extern int checkPosEqual(Pos p1, Pos p2);
40 extern int getDirection(Pos source, Pos destination);
41 extern int getDirectionLR(int ox, int x);
42 extern int getDirectionUD(int oy, int y);
43 extern string getDirectionChar(int direction);
44 extern int getReversedDirection(int direction);
45 extern int isConnected(Pos p1, Pos p2);
46 extern int isNext(Pos p1, Pos p2);
47 extern int isNextX(Pos p1, Pos p2);
48 extern int isNextY(Pos p1, Pos p2);

```



```
49 extern void myerror(int error_code);
50 extern void swapNum(int *num1, int *num2); // num1とnum2を交換
51 extern void surroundings(int *x, int *y, int direction);
52 extern Pos surroundings(Pos Data, int direction);
53 extern void swapPos(Pos *p1, Pos *p2);
54
55 #endif
```