List 1: main.cpp

```cpp
#include <iostream>
#include <vector>
// for OpenCV2
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>

// httpクライアント
#include "httpClient/client.h"
// 問題ヘッダ部(コメント部分)
#include "httpClient/QuestionHeader.hpp"

// 画像解析
#include "placement/PPMFILE.hpp"

// sort
#include "sort/PosData.h"
#include "sort/Process5.h"

// デバッグ用
#define VERBOSE

using namespace std;

// 問題解答関数
int solveProbrem(int id);

int main(){
 int id=1;

 // マルチプロセスとかしたい時のために
 solveProbrem(id);

 return EXIT_SUCCESS;
}

int solveProbrem(int id){
 ProkonClient client; // httpクライアント
 QuestionHeader header; // ヘッダデータ
 string res; // 受信データ
 string answer; // 解答データ
 // 受信した画像データ
 cv::Mat recievedData;
 // PPMFILE
 PPMFILE *img;
 // PosData
 PosData *data;
 Process5 *sort;

#ifdef VERBOSE
 cout << "Picture Downloading\nID: " << id<< endl;
#endif

 try {
  // 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)
  res=client.getProblem(id,header);
 } catch (runtime_error & exception) {
  cerr << "Exception: " << exception.what() << endl;
  exit(EXIT_FAILURE);
 }

#ifdef VERBOSE
 cerr << "Split X: " << header.splitX << endl;
 cerr << "Split Y: " << header.splitY << endl;
 cerr << "Selectable Count: " << header.selectableCount << endl;
 cerr << "Select Rate: " << header.selectRate << endl;
 cerr << "Exchange Rate: " << header.exchangeRate << endl;
#endif

 // Matにするために受け取った文字列をvector <char>型に変換
 vector<char> v(res.begin(),res.end());
 recievedData=cv::imdecode(cv::Mat(v),-1);

 data=new PosData(header.splitX,header.splitY);
 sort=new Process5(header.splitX,header.splitY);

#ifdef VERBOSE
 cout << "placement" << endl;
#endif
 // ここに画像解析処理
 // どこかでPosData設定が必要
 img= new PPMFILE(recievedData,header.splitX,header.splitY);
 img->calc_cost();
 // img->calc_cost_maru(); // まだこのコスト計算は実装されていない
 img->placement();
 // img->create_result_img(); //画像を表示しないなら使わないでOK
 img->set_PosData(data);

#ifdef VERBOSE
 cout << "sort" << endl;
#endif
 // ここにソート処理
 sort->importData(*data);

#ifdef VERBOSE
 cout <<"(not sort done)Answer And Send" << endl;
#endif

 try {
  // 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)
  res=client.sendAnswer(id,sort->sort());
 } catch (char const * exception) {

  cerr << "Exception: " << exception << endl;
  exit(EXIT_FAILURE);
 }
 cout << res << endl;

 return EXIT_SUCCESS;

}
```

List 2: httpClient/QuestionHeader.hpp

```
#ifndef __QUESTION_HEADER_H_INCLUDED__
#define __QUESTION_HEADER_H_INCLUDED__
// 問題ヘッダクラス
// 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
// 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
class QuestionHeader {
 public:
  // 分割数X,Y
  unsigned int splitX;
  unsigned int splitY;
  // レート(選択, 交換)
  unsigned int selectRate;
  unsigned int exchangeRate;
  // 交換可能回数
  unsigned int selectableCount;
};

#endif
```

List 3: httpClient/client.cpp

```
#include <curl/curl.h>
#include <iostream>
#include <sstream>
#include <string>
#include <vector>
// for isdigit
#include <cctype>

#include "client.h"
#include "QuestionHeader.hpp"

#define DEBUG 1

using namespace std;

// サーバー
const string ProkonClient::SERVER_ADDRESS="localhost/web2/pic";
// チーム固有トークン
const string ProkonClient::TEAM_TOKEN="SKYHIGH\nCHRONOS\nENDLESS\n";

// コンストラクタ
ProkonClient::ProkonClient(){
 curl = curl_easy_init();
 if(curl==NULL){
  throw runtime_error("CURL is NULL\n");
 }
}
// デストラクタ
ProkonClient::~ProkonClient(){
 curl_easy_cleanup(curl);
}

string ProkonClient::getProblem(int problemNo,QuestionHeader & data){
 // 受信データ
 string chunk;
 // 入れるポインタデータ
 vector<unsigned int *> dataPointer;
 // 現在操作するポインタ
 vector<unsigned int *>::iterator now;
 // フラグ
 bool enable;

 // 代入するデータを順に入れる
 dataPointer.push_back(& data.splitX);
 dataPointer.push_back(& data.splitY);
 dataPointer.push_back(& data.selectableCount);
 dataPointer.push_back(& data.selectRate);
 dataPointer.push_back(& data.exchangeRate);

 // URL用問題番号の文字列
 ostringstream probStr;
 //問題番号部分指定
 probStr.setf(ios::right);
 probStr.fill('0');
 probStr.width(2);
 probStr << problemNo;

 // URL
 string url="http://" + SERVER_ADDRESS + "/problem/prob" + probStr.str() + ".ppm"
    ;
#if DEBUG==1
 cout << url << endl;
#endif

 chunk=getData(url);
 // コメント抜き取り
 now=dataPointer.begin();
 **now=0;
 enable=false;
 for(char &c : chunk){
  // コメント読み取り有効
  if(c=='#'){
   // 有効
   enable=true;
  }
  // コメント読み取り無効
  if(c=='\n'){
   enable=false;
  }
  if(enable){
   if(isdigit(c)){
    **now= (**now) * 10+(c-'0');
   }else if(**now!=0){
    now++;
   }
  }
  if(now == dataPointer.end()){
   break;
  }
 }
 return chunk;
}

string ProkonClient::sendAnswer(int problemNo,string answer){
 // 送信データ
 string data;
 // 受信データ
 string chunk;
 //http://{ServerAddress}/SubmitAnswer
 string url="http://" + SERVER_ADDRESS + "/SubmitAnswer.pl";
 // エスケープされた文字列 free用
 char * escapedStr;

 data="playerid=";
 // チームトークン
 escapedStr=curl_easy_escape(curl,TEAM_TOKEN.c_str(),0);
 data+=escapedStr;
 curl_free(escapedStr);
 // 問題番号
 escapedStr=curl_easy_escape(curl,to_string(problemNo).c_str(),0);
 data+="&problemid="+(string)(escapedStr);
 curl_free(escapedStr);
 //回答
 escapedStr=curl_easy_escape(curl,answer.c_str(),0);
 data+="&answer="+(string)escapedStr;
 curl_free(escapedStr);

 chunk=getData(url,data);

 return chunk;
}

// postData: *直接送信する* (エスケープが必要)
string ProkonClient::getData(string url,string postData){
 CURLcode res;
 // httpコード (200とか404とか)
 long http_code=0;
 // ダウンロードしたデータ
 string chunk;

 // URL設定
 curl_easy_setopt(curl,CURLOPT_URL,url.c_str());
 // port:8080
 curl_easy_setopt(curl,CURLOPT_PORT,8080);
 // POSTデータ指定
 curl_easy_setopt(curl,CURLOPT_POSTFIELDS,postData.c_str());

 // 受信コールバック
 curl_easy_setopt(curl,CURLOPT_WRITEFUNCTION,ProkonClient::callbackWrite);
 // コールバックデータ書き込み先
 curl_easy_setopt(curl,CURLOPT_WRITEDATA,&chunk);
 // 実行
 res=curl_easy_perform(curl);
 // レスポンスコード受信
 curl_easy_getinfo(curl,CURLINFO_RESPONSE_CODE,&http_code);

 if(res != CURLE_OK){
  cerr << "!!!curl_easy_perform failed!!!" << endl;
  throw runtime_error("perform something wrong.");
 }
 if(http_code != 200){
  cerr << "STATUS CODE IS NOT 200:" << to_string(http_code) << endl;
  string msg="http code is "+to_string(http_code);
  throw runtime_error(msg);
 }
 return chunk;
}

string ProkonClient::getData(string url){
 return getData(url,"");
}

size_t ProkonClient::callbackWrite(char *ptr,size_t size, size_t nmemb,string *
    stream){
   // サイズ計算
   int dataLength = size * nmemb;
   // 書き込み
   stream->append(ptr,dataLength);
   // サイズを返す
   return dataLength;
  }
```

List 4: httpClient/client.h

```
#ifndef __PROKON_CLIENT_H_INCLUDED__
#define __PROKON_CLIENT_H_INCLUDED__

#include <string>
#include <stdio.h>
#include <fstream>
#include <curl/curl.h>

#include "QuestionHeader.hpp"

using namespace std;

class ProkonClient{
 private:
  // streamにsize*nmembの大きさのptrを追加
  // 大きさを返す
  static size_t callbackWrite(char *ptr,size_t size, size_t nmemb,string *stream
      );
  string getData(string url,string postData);
  string getData(string url);
  // curlオブジェクト
  CURL *curl;
 public:
  // コンストラクタ
  ProkonClient();
  ~ProkonClient();
  static const string SERVER_ADDRESS;
  static const string TEAM_TOKEN;
  string getProblem(int problemNo,QuestionHeader & data);
  string sendAnswer(int problemNo,string answer);
};
#endif
```

List 5: placement/PPMFILE.cpp

```
#include <iostream>

#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>

#include "PPMFILE.hpp"
#include "../sort/PosData.h"

// tupleを比較するときのルール
bool my_compare( const COST_TUPLE &lhs, const COST_TUPLE &rhs){
        if (std::get<0>(lhs) != std::get<0>(rhs)) return std::get<0>(lhs) < std::get<0>(rhs);
        if (std::get<1>(lhs) != std::get<1>(rhs)) return std::get<1>(lhs) < std::get<1>(rhs);
        if (std::get<2>(lhs) != std::get<2>(rhs)) return std::get<2>(lhs) < std::get<2>(rhs);
        return std::get<2>(lhs) < std::get<2>(rhs);
}

// 方向に対するpairを作る
pair<int,int> make_direction_pair(int direction){
        pair<int,int> dire_pair;
        switch(direction){
                case DIRE_U:
                        dire_pair = make_pair(0,-1);
                        break;
                case DIRE_D:
                        dire_pair = make_pair(0,1);
                        break;
                case DIRE_R:
                        dire_pair = make_pair(1,0);
                        break;
                case DIRE_L:
                        dire_pair = make_pair(-1,0);
                        break;
        }

        return dire_pair;
}

// 方向を逆にする
int inverse_direction(int dire){
        int inv_dire;
        switch(dire){
                case DIRE_U:
                        inv_dire = DIRE_D;
                        break;
                case DIRE_D:
                        inv_dire = DIRE_U;
                        break;
                case DIRE_R:
                        inv_dire = DIRE_L;
                        break;
                case DIRE_L:
                        inv_dire = DIRE_R;
                        break;
                        default:
                                cout << "error direction" << endl;
                                break;
        }
        return inv_dire;
}

PPMFILE::PPMFILE(cv::Mat origin_img_tmp, int piece_x, int piece_y){
        origin_img = origin_img_tmp.clone();
        part_size_x = piece_x;
        part_size_y = piece_y;

        //それぞれのピースに分割
        this->create_partition();
}

void PPMFILE::disp_img(int type){
        cv::namedWindow("image", CV_WINDOW_AUTOSIZE | CV_WINDOW_FREERATIO);
        switch (type){
                case ORIGIN_IMG:        cv::imshow("image", origin_img);
                                                        break;
                case LINE_IMG: cv::imshow("image", line_img);
                                                break;
                case RESULT_IMG:        cv::imshow("image", result_img);
                                                        break;
                default:
                                                        break;
        }
}

void PPMFILE::write_line(void){
        line_img = origin_img.clone(); // origin_img をコピー
        //縦向きの線
        for(int x=1; x < part_size_x; x++){
                cv::line( line_img, cv::Point( x*(line_img.cols/part_size_x), 0),
                     cv::Point( x*(line_img.cols/part_size_x), line_img.rows), cv
                    ::Scalar( 200, 0, 0), 2, 0);
        }
        //横向きの線
        for(int y=1; y < part_size_y; y++){
                cv::line( line_img, cv::Point( 0, y*(line_img.rows/part_size_y)),
                     cv::Point( line_img.cols, y*(line_img.rows/part_size_y)), cv
                    ::Scalar( 0, 0, 200), 2, 0);
        }
}

void PPMFILE::create_partition(void){
        int part_width = origin_img.cols/part_size_x;
        int part_height = origin_img.rows/part_size_y;
        //pair_img[n]にそれぞれのピースを代入
        for(int y=0; y < part_size_y; y++){
                for(int x=0; x < part_size_x; x++){
                        cv::Mat tmp(origin_img, cv::Rect(x*part_width, y*
                            part_height, part_width, part_height));
                        part_img.push_back(tmp);
                }
        }
}

void PPMFILE::calc_cost(void){
        // costのサイズ変更
        cost.resize( part_size_x*part_size_y);
        for(int i=0; i<part_size_x*part_size_y; i++){
                cost[i].resize(4);
                for(int j=0; j<4; j++){
                        cost[i][j].resize(part_size_x*part_size_y);
                }
```

```cpp
        }
        cout << "start_calc" << endl;
        int cost_tmp[4];
        // 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
        for(int abs_xy=0; abs_xy < part_size_x * part_size_y; ++abs_xy){
                for(int xy=abs_xy+1; xy < part_size_x * part_size_y; ++xy){
                        //コスト初期化
                        for(int i=0; i<4; i++){
                                cost_tmp[i] = 0;
                        }
                        //カラーの数だけ繰り返す
                        for(int c=0; c < part_img[abs_xy].channels(); ++c){
                                //各ピクセル上下
                                for(int px_x=0; px_x < part_img[0].cols; ++px_x){
                                        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]);
                                        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]);
                                }
                                //各ピクセル左右
                                for(int px_y=0; px_y < part_img[0].rows; ++px_y){
                                        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]);
                                        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]);
                                }
                        }
                        // 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
                        // を×る事によって、結合度の重みを長さに依存させない
                        // ワンちゃんオーバーフローが恐い(たぶん大丈夫)
                        cost_tmp[DIRE_U] *= part_img[0].rows;
                        cost_tmp[DIRE_D] *= part_img[0].rows;
                        cost_tmp[DIRE_R] *= part_img[0].cols;
                        cost_tmp[DIRE_L] *= part_img[0].cols;
                        // cost_t [ コスト, 自分の座標, 相手の座標, 方向]
                        // cost [自分の座標][方向][相手の座標]
                        for(int k=0; k < 4; ++k){
                                cost_t.push_back(make_tuple( cost_tmp[k], abs_xy, xy, k));
                                cost[abs_xy][k][xy] = make_pair( cost_tmp[k], xy );
                        }
                }
        }
        cout << "start_sort" << endl;
        // 独自のルールでCOST_TUPLEをソート
        sort( cost_t.begin(), cost_t.end(), my_compare);
        cout << "end calc" << endl;
}

void PPMFILE::disp_cost_list(void){
        // 表示
        cout << "score : my_pos : pair_pos : direction " << endl;
        for(int i=0; i < cost_t.size(); i++){
                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;
        }
}

void PPMFILE::placement(void){
        // 使われたかどうかのフラグ
        vector<SCRAP> scraps;
        vector<int> used_part(part_size_x*part_size_y, -1);
        // 目的の場所への差分
        int dif_x, dif_y;
        int part_s, part_l;
        int part_s_x, part_s_y;
        int part_l_x, part_l_y;
        int part_1, part_2;
        int part_1_x, part_1_y;
        int part_2_x, part_2_y;
        // 再配置の時に使う
        int pos_x_min = 0;
        int pos_y_min = 0;
        for(int i=0; i < cost_t.size(); i++){
                SCRAP scrap_tmp;
                part_1 = used_part[get<1>(cost_t[i])];
                part_2 = used_part[get<2>(cost_t[i])];

                if(part_1 == -1){
                        if(part_2 == -1){ // まだ使われてなかったら
                                // どちらもまだ使われていない
                                used_part[get<1>(cost_t[i])] = used_part[get<2>(cost_t[i])] = scraps.size();     // 追加したscrap番号を保持
                                scrap_tmp.elements[get<1>(cost_t[i])] = make_pair( 0, 0);
                                scrap_tmp.used_p[make_pair(0,0)] = get<1>(cost_t[i]);
                                scrap_tmp.elements[get<2>(cost_t[i])] = make_direction_pair(get<3>(cost_t[i]));
                                scrap_tmp.used_p[make_direction_pair(get<3>(cost_t[i]))] = get<2>(cost_t[i]);
                                scraps.push_back(scrap_tmp);
                        }
                        else{                                                   // 2だけつかわれてる
                                part_2_x = scraps[part_2].elements[get<2>(cost_t[i])].first;
                                part_2_y = scraps[part_2].elements[get<2>(cost_t[i])].second;
                                // 目的の場所(2が基準だから左右が逆になる)
                                switch(get<3>(cost_t[i])){
                                        case DIRE_U:
                                                ++part_2_y;
                                                break;
                                        case DIRE_D:
                                                --part_2_y;
                                                break;
                                        case DIRE_R:
                                                --part_2_x;
                                                break;
                                        case DIRE_L:
                                                ++part_2_x;
                                                break;
                                }
                                if(scraps[part_2].used_p.find(make_pair(part_2_x, part_2_y)) == scraps[part_2].used_p.end()){
                                        used_part[get<1>(cost_t[i])] = part_2;
                                        scraps[part_2].elements[get<1>(cost_t[i])] = make_pair( part_2_x, part_2_y);
                                        scraps[part_2].used_p[make_pair( part_2_x, part_2_y)] = get<1>(cost_t[i]);
                }else{
                    // すでに要素が入っていた場合
                }
            }
        }else{
            if(part_2 == -1){       // 1だけ使われてる
                part_1_x = scraps[part_1].elements[get<1>(cost_t[i])].first;
                part_1_y = scraps[part_1].elements[get<1>(cost_t[i])].second;
                // 目的の場所
                switch(get<3>(cost_t[i])){
                    case DIRE_U:
                        --part_1_y;
                        break;
                    case DIRE_D:
                        ++part_1_y;
                        break;
                    case DIRE_R:
                        ++part_1_x;
                        break;
                    case DIRE_L:
                        --part_1_x;
                        break;
                }
                // 目的の場所に要素が入っていなかった場合
                if(scraps[part_1].used_p.find(make_pair(part_1_x, part_1_y)) == scraps[part_1].used_p.end()){
                    used_part[get<2>(cost_t[i])] = part_1;
                    scraps[part_1].elements[get<2>(cost_t[i])] = make_pair( part_1_x, part_1_y);
                    scraps[part_1].used_p[make_pair( part_1_x, part_1_y)] = get<2>(cost_t[i]);
                }else{
                    // 既に要素が入っていた場合
                }
            }else{  // 両方ともつかわれている -> スクラップの添字が小さい方に大きい方をくっつける
                if( part_1 != part_2){  // 同じスクラップでない場合
                    part_s = MIN_2( part_1, part_2);
                    part_l = BIG_2( part_1, part_2);

#ifdef DEBUG
                    cout << "now data " << endl;
                    for(int i=0; i < scraps.size(); i++){
                        cout << "pair : " << i << endl;
                        for(map<int, pair<int,int> >::iterator j = scraps[i].elements.begin(); j != scraps[i].elements.end(); j++){
                            int key = j->first;
                            pair<int, int> pos = j->second;
                            cout << "  (" << CONV_X(key) << "," << CONV_Y(key) << ") || (" << pos.first << "," << pos.second << ")" << endl;
                        }
                    }
                    cout << "part_1 : " << part_1 << endl;
                    cout << "marg : " << part_s << " to " << part_l << endl;
                    cout << "PP1 : (" << CONV_X(get<1>(cost_t[i])) << "," << CONV_Y(get<1>(cost_t[i
					])) << ")" << endl;
					cout << "PP2 : (" << CONV_X(get<2>(cost_t[i])) << "," << CONV_Y(get<2>(cost_t[i])) << ")" << endl;
					cout << "discription : " << get<3>(cost_t[i]) << endl;
#endif
					if(part_s == part_1){
						cout << "type:A" << endl;
						part_s_x = scraps[part_1].elements[get<1>(cost_t[i])].first;
						part_s_y = scraps[part_1].elements[get<1>(cost_t[i])].second;
						part_l_x = scraps[part_2].elements[get<2>(cost_t[i])].first;
						part_l_y = scraps[part_2].elements[get<2>(cost_t[i])].second;
						cout << "p_s_x : " << part_s_x << " p_s_y : " << part_s_y << endl;
						switch(get<3>(cost_t[i])){
							case DIRE_U:
								--part_s_y;
								break;
							case DIRE_D:
								++part_s_y;
								break;
							case DIRE_R:
								++part_s_x;
								break;
							case DIRE_L:
								--part_s_x;
								break;
						}
					}else{
						// part_2 を基準とする
						cout << "type:B" << endl;
						part_l_x = scraps[part_1].elements[get<1>(cost_t[i])].first;
						part_l_y = scraps[part_1].elements[get<1>(cost_t[i])].second;
						part_s_x = scraps[part_2].elements[get<2>(cost_t[i])].first;
						part_s_y = scraps[part_2].elements[get<2>(cost_t[i])].second;
						switch(get<3>(cost_t[i])){
							case DIRE_U:
								++part_s_y;
								break;
							case DIRE_D:
								--part_s_y;
								break;
							case DIRE_R:
								--part_s_x;
								break;
							case DIRE_L:
								++part_s_x;
								break;
						}
```

```
                    }
                    cout << "p_s_x : " << part_s_x << " p_s_y : " << part_s_y << endl;
                    dif_x = part_s_x - part_l_x;
                    dif_y = part_s_y - part_l_y;
                    cout << "dif_x : " << dif_x << " dif_y : " << dif_y << endl;
                    // scrapが小さい方に統合(何も考えずに)
                    for(map<int, pair<int,int> >::iterator j = scraps[part_l].elements.begin(); j != scraps[part_l].elements.end(); j++){
                        int key = j->first;
                        used_part[key] = part_s;
                        pair<int, int> pos = j->second;
                        scraps[part_s].elements[key] = make_pair( pos.first + dif_x, pos.second + dif_y);
                        scraps[part_s].used_p[make_pair(pos.first + dif_x, pos.second + dif_y)] = key;
                    }
                    scraps[part_l].elements.clear();
                    for(int i=0; i < scraps.size(); i++){
                        cout << "pair : " << i << endl;
                        for(map<int, pair<int,int> >::iterator j = scraps[i].elements.begin(); j != scraps[i].elements.end(); j++){
                            int key = j->first;
                            pair<int, int> pos = j->second;
                            cout << "  (" << CONV_X(key) << "," << CONV_Y(key) << ") || (" << pos.first << "," << pos.second << ")" << endl;
                        }
                    }
                }
            }
        }
    }
    // 座標の左上を0,0にする
    // この時点でscrapsは[0]しか要素を持たないはず
    for(map<int, pair<int,int> >::iterator j = scraps[0].elements.begin(); j != scraps[0].elements.end(); j++){
        int key = j->first;
        pair<int, int> pos = j->second;
        if(pos.first < pos_x_min)
            pos_x_min = pos.first;
        if(pos.second < pos_y_min)
            pos_y_min = pos.second;
    }
    // 一番小さいものに座標を合わせて再配置
    for(map<int, pair<int,int> >::iterator j = scraps[0].elements.begin(); j != scraps[0].elements.end(); j++){
        int key = j->first;
        pair<int, int> pos = j->second;
        scraps[0].elements[key] = make_pair( (pos.first - pos_x_min), (pos.second - pos_y_min));
    }
    // グローバルに結果をコピー
    placement_pos = scraps[0].elements;
}

```

```cpp
void PPMFILE::disp_placement(void){
        cout << "----------" << endl;
        cout << "placement " << endl;
        for(map<int, pair<int,int> >::iterator j = placement_pos.begin(); j != placement_pos.end(); j++){
                int key = j->first;
                pair<int, int> pos = j->second;
                cout << "  (" << CONV_X(key) << "," << CONV_Y(key) << ") || (" << pos.first << "," << pos.second << ") " << endl;
        }
}

void PPMFILE::create_result_img(void){
        int part_width = origin_img.cols/part_size_x;
        int part_height = origin_img.rows/part_size_y;

        int max_part_x = 0, max_part_y = 0;

        // 最大の幅と高さを求める
        for(map<int, pair<int,int> >::iterator j = placement_pos.begin(); j != placement_pos.end(); j++){
                int key = j->first;
                pair<int, int> pos = j->second;
                if(max_part_x < pos.first)
                        max_part_x = pos.first;
                if(max_part_y < pos.second)
                        max_part_y = pos.second;
        }
        cout << "mpx :" << max_part_x << " mpy :" << max_part_y << endl;

        // 大きさに応じたサイズのresult_imgを作成
        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));
        // cv::Mat tmp_result_img( cv::Size( 2000, 2000), CV_8UC3, cv::Scalar(0,0,0));

        for(map<int, pair<int,int> >::iterator j = placement_pos.begin(); j != placement_pos.end(); j++){
                int key = j->first;
                pair<int, int> pos = j->second;
                part_img[key].copyTo(tmp_result_img( \
                                        cv::Rect( \
                                                pos.first*part_width, \
                                                pos.second*part_height, \
                                                part_width, \
                                                part_height) \
                                        ));
                cout << "s_x :" << pos.first*part_width << " s_y :" <<  pos.second*part_height << " CopyTo:" << key << endl;
        }
        result_img = tmp_result_img.clone();
}

int PPMFILE::get_cost(int xy_1, int xy_2, int dire){
        int pair_cost;
        if(xy_1 < xy_2){
                pair_cost = cost[xy_1][dire][xy_2].first;
        }else if(xy_1 > xy_2){
                // 比較順が変わると上下左右が変わる
                dire = inverse_direction(dire);
                pair_cost = cost[xy_2][dire][xy_1].first;
        }else{
                pair_cost = -1;
        }
        return pair_cost;
}

```

```
void PPMFILE::set_PosData(PosData *data){
        for(map<int, pair<int,int> >::iterator j = placement_pos.begin(); j != placement_pos.end(); j++){
                int key = j->first;
                pair<int, int> pos = j->second;

                data->setData( CONV_X(key), CONV_Y(key), pos.first, pos.second);
        }
}


void PPMFILE::calc_cost_maru(void){
        // costのサイズ変更
        cost_maru.resize( part_size_x*part_size_y);
        for(int i=0; i<part_size_x*part_size_y; i++){
                cost_maru[i].resize(4);
                for(int j=0; j<4; j++){
                        cost_maru[i][j].resize(part_size_x*part_size_y);
                }
        }

        cout << "start_calc_maru" << endl;

        int cost_tmp[4];
        // 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
        for(int abs_xy=0; abs_xy < part_size_x * part_size_y; ++abs_xy){
                for(int xy=abs_xy+1; xy < part_size_x * part_size_y; ++xy){
                        //コスト初期化
                        for(int i=0; i<4; i++){
                                cost_tmp[i] = 0;
                        }
                        //カラーの数だけ繰り返す
                        for(int c=1; c < part_img[abs_xy].channels(); ++c){
                                //各ピクセル上下
                                for(int px_x=0; px_x < part_img[0].cols; ++px_x){
                                        cost_tmp[DIRE_U] += abs(part_img[abs_xy].at<cv::Vec3b>( 0, px_x)[c] - (part_img[abs_xy].at<cv::Vec3b>( 1, px_x)[c] - part_img[xy].at<cv::Vec3b>( part_img[0].rows-1, px_x)[c])/2);
                                        cost_tmp[DIRE_D] += abs(part_img[abs_xy].at<cv::Vec3b>( part_img[0].rows-1, px_x)[c] - (part_img[abs_xy].at<cv::Vec3b>( 0, px_x)[c] - part_img[xy].at<cv::Vec3b>( part_img[0].rows-2, px_x)[c])/2);
                                }
                                //各ピクセル左右
                                for(int px_y=0; px_y < part_img[0].rows; ++px_y){
                                        // cost_tmp[DIRE_R] += abs(part_img[abs_xy].at<cv::Vec3b>( px_y, part_img[0].cols-1)[c] - (part_img[abs_xy].at<cv::Vec3b>( px_y, 0)[c] - part_img[xy].at<cv::Vec3b>( px_y, part_img[0].cols-2)[c])/2);


                                        //      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]);
                                        // 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]);
                                }
                        }

                        // 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
                        // を×る事によって、結合度の重みを長さに依存させない
                        // ワンちゃんオーバーフローが恐い(たぶん大丈夫)
                        cost_tmp[DIRE_U] *= part_img[0].rows;
                        cost_tmp[DIRE_D] *= part_img[0].rows;
                        cost_tmp[DIRE_R] *= part_img[0].cols;
                        cost_tmp[DIRE_L] *= part_img[0].cols;

                        // cost_t [ コスト, 自分の座標, 相手の座標, 方向]
                        // cost [自分の座標][方向][相手の座標]
                        for(int k=0; k < 4; ++k){
                                cost_t_maru.push_back(make_tuple( cost_tmp[k], abs_xy, xy, k));
                                cost_maru[abs_xy][k][xy] = make_pair( cost_tmp[k], xy);
                        }
                }
        }

        cout << "start_sort" << endl;
        // 独自のルールでCOST_TUPLEをソート
        sort( cost_t.begin(), cost_t.end(), my_compare);
        cout << "end calc" << endl;
}
```

List 6: placement/PPMFILE.hpp

```
#include <vector>
#include <tuple>

#include <opencv2/core/core.hpp>

#include "../sort/PosData.h"
#define DEBUG

#define ORIGIN_IMG 0
#define LINE_IMG 1
#define RESULT_IMG 2

#define DIRE_U 0
#define DIRE_D 1
#define DIRE_R 2
#define DIRE_L 3


#define MIN_2( A, B) ((A) < (B) ? (A) : (B))
#define BIG_2( A, B) ((A) > (B) ? (A) : (B))

using namespace std;

typedef tuple< int, int, int, int> COST_TUPLE;

// スクラップ(いくつかのパーツの集まり)
typedef struct{
        map<int,pair<int,int> > elements;       // パーツidから、それが使われているか?
        map<pair<int, int>, int> used_p;// ポジションから、そこに何があるか?
}SCRAP;


class PPMFILE{
        private:
                int part_size_x, part_size_y;   //左右どれだけのピース数があるか?

```

```
			cv::Mat origin_img;
			cv::Mat line_img;
			cv::Mat result_img;

			vector<cv::Mat> part_img;

			// cost配列(3次元)
			vector< vector< vector< pair<int,int> > > > cost;
			vector< vector< vector< pair<int,int> > > > cost_maru;
			// cost_t配列(1次元)
			vector<COST_TUPLE> cost_t;
			vector<COST_TUPLE> cost_t_maru;
			// 配置の位置を示す配列
			map<int,pair<int,int> > placement_pos;

		public:

			// 2次元を1次元に
			inline int CONV_XY(int x,int y){
				return x+y * part_size_x;
			}
			// XY座標を1次元に
			inline int CONV_X(int XY){
				return XY % part_size_x;
			}
			inline int CONV_Y(int XY){
				return XY/part_size_x;
			}
			PPMFILE(cv::Mat origin_img_tmp, int piece_x, int piece_y);
			//画像を表示する
			void disp_img(int type);
			//境界線を引く
			void write_line(void);
			// それぞれのピースに分割
			void create_partition(void);
			// 配列の近似値を計算nする。
			void calc_cost(void);
			// まるさん方式
			void calc_cost_maru(void);

			void disp_cost_list(void);

			// 配置
			void placement(void);

			void disp_placement(void);
			//PosDataにデータを挿入
			void set_PosData(PosData *data);

			// 配置してみた時の表示
			void create_result_img(void);

			// xy_1のdire方向のxy_2とのcost
			int get_cost(int xy_1, int xy_2, int dire);
};
```

List 7: placement/placement.cpp

```
#include <iostream>
#include <fstream>
#include <string>

#include <stdlib.h> /* abs 絶対値 */
#include <stdlib.h> /* rand() */
#include <time.h>   /* time */

```

```
#include <OpenGL/gl.h>
#include <GLUT/glut.h>

#define FILENAME "1.ppm"
#define PIECE_X 16
#define PIECE_Y 16

using namespace std;


void display(void);
void init(void);
void resize( int, int);
void Point( int, int, int, int, int, int);

class PPMFILE{
        private:
                int width, height, bright;

                //image[col][x][y] (メモリの動的確保)
                unsigned char ***image = new unsigned char **[3];
        public:
                void init_data(string filename){
                        // バイナリモードで開く
                        ifstream fin;
                        fin.open(filename, ios::in | ios::binary);

                        if(!fin){
                                cout << "file : \"" << filename << "\"reading failure." << endl;
                                exit(1);
                        }else{
                                cout << "file : \"" << filename << "\" reading succes." << endl;
                        }

                        // 1行目の取得 P6:バイナリモード or P3:テキストモード
                        string line;
                        getline( fin, line);
                        cout << "format : " << line << endl;
                        // 3行目(width height の取得)
                        while (width == 0 || height == 0){
                                string line;
                                getline( fin,line);
                                if(line.at(0) !='#')
                                        sscanf(line.c_str(),"%d %d",&width, &height);
                        }
                        cout << "width: " << width << endl << "height: " << height << endl;

                        //問題文形式だと、輝度で1行使っている
                        getline( fin, line);
                        sscanf(line.c_str(),"%d",&bright);
                        cout << "bright: " << bright << endl;

                        //動的画像保存領域の確保
                        for(int i=0; i<3; ++i){
                                image[i] = new unsigned char*[width];
                                for(int x=0; x<width; ++x){
                                        image[i][x] = new unsigned char[height];
                                }
                        }

                        //画像の読み込み
                        for(int y=0; y<height; ++y){
                                for(int x=0; x<width; ++x){
```

```cpp
                                                for(int col=0; col<3; ++col){
                                                        fin.read(reinterpret_cast<char*>(&image[col][x][y]),1);
                                                }
                                        }
                                }

                                fin.close();
                        }

                        void delete_data(){
                                for(int i=0; i<3; ++i){
                                        for(int x=0; x<width; ++x){
                                                delete[] image[i][x];
                                        }
                                        delete[] image[i];
                                }
                                delete[] image;
                                cout << "create delete memory" << endl;
                        }

                        void disp_data(){
                                for(int y=0; y<height; ++y){
                                        for(int x=0; x<width; ++x){
                                                Point( x, y, 1, image[0][x][y],image[1][x][y],image[2][x][y]);
                                                //赤だけ表示とか
                                                //Point( x, y, 1, image[0][x][y],0,0);
                                        }
                                }
                        }

                        //画面サイズをwidth,heightで初期化する
                        void imageGlutInitWindowSize(){
                                glutInitWindowSize( width, height);
                        }

                        int get_width(){
                                return width;
                        }
                        int get_height(){
                                return height;
                        }

                        void set_data(int x, int y, int col, unsigned char data){
                                image[col][x][y] = data;
                        }
                        int get_data(int x, int y, int col){
                                return image[col][x][y];
                        }
};

class PIECE_DATA{
        private:
                int piece_width, piece_height;
                unsigned char ***piece = new unsigned char **[3];

        public:
                int rx, ry;
                int lx, ly;
                int ux, uy;
                int dx, dy;
                double rp, lp, up, dp;

                int re_pos_x, re_pos_y;

                void init_data(int width, int height);
                void set_data(int x, int y, int col, unsigned char data);
                int get_data(int x, int y, int col);
                void delete_data();
                void disp_data_pos(int , int);
};

void PIECE_DATA::init_data(int width, int height){
        piece_width = width;
        piece_height = height;

        //動的画像保存領域の確保
        for(int i=0; i<3; ++i){
                piece[i] = new unsigned char*[piece_width];
                for(int x=0; x<width; ++x){
                        piece[i][x] = new unsigned char[piece_height];
                }
        }
}
void PIECE_DATA::set_data(int x, int y, int col, unsigned char data){
        piece[col][x][y] = data;
}
int PIECE_DATA::get_data(int x, int y, int col){
        return piece[col][x][y];
}
void PIECE_DATA::delete_data(){
        for(int i=0; i<3; ++i){
                for(int x=0; x<piece_width; ++x){
                        delete[] piece[i][x];
                }
                delete[] piece[i];
        }
        delete[] piece;
}
void PIECE_DATA::disp_data_pos(int start_x, int start_y){
        for (int y = 0; y < piece_height; y++) {
                for (int x = 0; x < piece_width; x++) {
                        Point( start_x+x, start_y+y, 1, piece[0][x][y], piece[1][
                            x][y], piece[2][x][y]);
                }
        }
}

PPMFILE pic_data;
PIECE_DATA **piece;

int piece_x;
int piece_y;

int piece_width, piece_height;

//OpenGL
void display(void)
{
        glClear(GL_COLOR_BUFFER_BIT);
        pic_data.disp_data();

        for(int pos_y=0; pos_y<piece_y; pos_y++){
                for(int pos_x=0; pos_x<piece_x; pos_x++){
                        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);
                        cout << pos_x << ":" << pos_y << "|" << piece[pos_x][
                            pos_y].re_pos_x << ":" << piece[pos_x][pos_y].re_pos_y
                             << endl;
                }
        }
        glFlush();
```

```
}

//どれだけ隣り合うドット同士の色が近いのか
//数字分けはとりあえず適当
int compare_dot(int a, int b){
        if (a==b)
                return 10;
        else if(abs(a-b) <= 5)
                return 9;
        else if(abs(a-b) <=10)
                return 4;
        else if(abs(a-b) <=30)
                return 2;
        else
                return 0;
}


int main(int argc, char *argv[])
{
        //乱数初期化
        srand((unsigned)time(NULL));

        pic_data.init_data(FILENAME);

        //分割数が入る予定
        piece_x = PIECE_X;
        piece_y = PIECE_Y;

        //画像の大きさ
        int image_width = pic_data.get_width();
        int image_height = pic_data.get_height();

        //分割画像の大きさ
        piece_width = image_width/piece_x;
        piece_height = image_height/piece_y;

        //各ピースの配列宣言
        piece = new PIECE_DATA *[piece_x];
        for(int i=0; i<piece_x; i++){
                piece[i] = new PIECE_DATA[piece_y];
        }

        //各ピースのメモリの確保
        for(int i=0; i<piece_x; i++){
                for(int j=0; j<piece_y; j++){
                        piece[i][j].init_data( piece_width, piece_height);
                }
        }

        //ピースごとに画像を取得
        for(int def_y=0; def_y<piece_y; def_y++){
                for(int def_x=0; def_x<piece_x; def_x++){

                        for(int y=0; y<piece_height; y++){
                                for(int x=0; x<piece_width; x++){
                                        for(int col=0; col<3; col++){
                                                piece[def_x][def_y].set_data( x,
                                                    y, col, pic_data.get_data( x+(
                                                    piece_width*def_x), y+(
                                                    piece_height*def_y), col));
                                        }
                                }
                        }

                }
        }

        //比較結果保存 メ モ リ 確 保
        int **compare_r = new int *[piece_x];
        int **compare_l = new int *[piece_x];
        int **compare_u = new int *[piece_x];
        int **compare_d = new int *[piece_x];
        for(int i=0; i<piece_x; i++){
                compare_r[i] = new int[piece_y];
                compare_l[i] = new int[piece_y];
                compare_u[i] = new int[piece_y];
                compare_d[i] = new int[piece_y];
        }

        //自分の見ているピース。調べたい中心

        int max_rx, max_ry, max_r;
        int max_lx, max_ly, max_l;
        int max_ux, max_uy, max_u;
        int max_dx, max_dy, max_d;

        //自分の見るピース
        for(int pos_y=0; pos_y<piece_y; pos_y++){
                for(int pos_x=0; pos_x<piece_x; pos_x++){

                        //初期化
                        for(int i=0; i<piece_x; i++){
                                for(int j=0; j<piece_y; j++){
                                        compare_r[i][j] = 0;
                                        compare_l[i][j] = 0;
                                        compare_u[i][j] = 0;
                                        compare_d[i][j] = 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++){

                                        for(int y=0; y<piece_height; y++){
                                                //画像の右側
                                                compare_r[def_x][def_y] += compare_dot( piece[pos_x][pos_y].get_data( piece_width-1, y, 0), piece[def_x][def_y].get_data( 0, y, 0));
                                                compare_r[def_x][def_y] += compare_dot( piece[pos_x][pos_y].get_data( piece_width-1, y, 1), piece[def_x][def_y].get_data( 0, y, 1));
                                                compare_r[def_x][def_y] += compare_dot( piece[pos_x][pos_y].get_data( piece_width-1, y, 2), piece[def_x][def_y].get_data( 0, y, 2));

                                                //画像の左側
                                                compare_l[def_x][def_y] += compare_dot( piece[pos_x][pos_y].get_data( 0, y, 0), piece[def_x][def_y].get_data( piece_width-1, y, 0));
                                                compare_l[def_x][def_y] += compare_dot( piece[pos_x][pos_y].get_data( 0, y, 1), piece[def_x][def_y].get_data( piece_width-1, y, 1));
                                                compare_l[def_x][def_y] += compare_dot( piece[pos_x][pos_y].get_data( 0, y, 2), piece[def_x][def_y].get_data( piece_width-1, y, 2));
                                        }
                                        for(int x=0; x<piece_width; x++){
                                                //画像の上側
                                                compare_u[def_x][def_y] += compare_dot( piece[pos_x][pos_y].get_data( x, 0, 0), piece[def_x][def_y].get_data( 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][pos_y].get_data( x, piece_height-1, 0), piece[def_x][def_y].get_data( x, 0, 0));
                                                compare_d[def_x][def_y] += compare_dot( piece[pos_x][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][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];
                                                max_lx = def_x;
                                                max_ly = def_y;
                                        }
                                        if(compare_u[def_x][def_y] > max_u){
                                                max_u = compare_u[def_x][def_y];
                                                max_ux = def_x;
                                                max_uy = def_y;
                                        }
                                        if(compare_d[def_x][def_y] > max_d){
                                                max_d = compare_d[def_x][def_y];
                                                max_dx = def_x;
                                                max_dy = def_y;
                                        }
                                }
                        }

                        piece[pos_x][pos_y].rx = max_rx;
                        piece[pos_x][pos_y].ry = max_ry;
                        piece[pos_x][pos_y].rp = (max_r)/(double)(piece_height*3*10)*100;
                        //cout << max_r << ":aaa" << endl;

                        piece[pos_x][pos_y].lx = max_lx;
                        piece[pos_x][pos_y].ly = max_ly;
                        piece[pos_x][pos_y].lp = (max_l)/(double)(piece_height*3*10)*100;

                        piece[pos_x][pos_y].ux = max_ux;
                        piece[pos_x][pos_y].uy = max_uy;
                        piece[pos_x][pos_y].up = (max_u)/(double)(piece_width*3*10)*100;

                        piece[pos_x][pos_y].dx = max_dx;
                        piece[pos_x][pos_y].dy = max_dy;
                        piece[pos_x][pos_y].dp = (max_d)/(double)(piece_width*3*10)*100;

                    }
                }

                //結果表示
                for(int pos_y=0; pos_y<piece_y; pos_y++){
                    for(int pos_x=0; pos_x<piece_x; pos_x++){
                        cout << "[" << pos_x << ":" << pos_y << "]" << endl;
                        cout << "R: (" << piece[pos_x][pos_y].rx << ":" << piece[pos_x][pos_y].ry << ")" << "[" << piece[pos_x][pos_y].rp << "]" << endl;
                        cout << "L: (" << piece[pos_x][pos_y].lx << ":" << piece[pos_x][pos_y].ly << ")" << "[" << piece[pos_x][pos_y].lp << "]" << endl;
                        cout << "U: (" << piece[pos_x][pos_y].ux << ":" << piece[pos_x][pos_y].uy << ")" << "[" << piece[pos_x][pos_y].up << "]" << endl;
                        cout << "D: (" << piece[pos_x][pos_y].dx << ":" << piece[pos_x][pos_y].dy << ")" << "[" << piece[pos_x][pos_y].dp << "]" << endl;
                    }
                }


                //ここから並び替え
                int datum_x, datum_y;

                int r1p, r2p;
                int r1x,r1y, r2x,r2y;

                do{
                    //適当に基準となるピースを選択
                    datum_x = rand()%piece_x;
                    datum_y = rand()%piece_y;

                    cout << "dx: " << datum_x << " dy: " << datum_y << endl;

                    //自分の一つ右のピースの
                    r1p = piece[datum_x][datum_y].rp;
                    r1x = piece[piece[datum_x][datum_y].rx][piece[datum_x][datum_y].ry].dx;
                    r1y = piece[piece[datum_x][datum_y].rx][piece[datum_x][datum_y].ry].dy;

                    r2p = piece[datum_x][datum_y].dp;
                    r2x = piece[piece[datum_x][datum_y].dx][piece[datum_x][datum_y].dy].rx;
                    r2y = piece[piece[datum_x][datum_y].dx][piece[datum_x][datum_y].dy].ry;
                }while(r1x != r2x || r1y != r2y || r1p <= 50 || r2p <= 50);

                //まずは、左に行けるところまで
                do{
                    cout << "dx: " << datum_x << " dy: " << datum_y << endl;

                    r1p = piece[datum_x][datum_y].lp;
                    r1x = piece[piece[datum_x][datum_y].lx][piece[datum_x][datum_y].ly].dx;
                    r1y = piece[piece[datum_x][datum_y].lx][piece[datum_x][datum_y].ly].dy;

                    r2p = piece[piece[datum_x][datum_y].dx][piece[datum_x][datum_y].dy].lp;
                    r2x = piece[piece[datum_x][datum_y].dx][piece[datum_x][datum_y].dy].lx;
                    r2y = piece[piece[datum_x][datum_y].dx][piece[datum_x][datum_y].dy].ly;

                    if((r1x == r2x) && (r1y == r2y) && (r1p >= 50 || r2p >= 50 )){
                        cout << "left" << endl;
                        int tmp;
                        tmp = datum_x;
                        datum_x = piece[datum_x][datum_y].lx;
                        datum_y = piece[tmp][datum_y].ly;
                    }else{
                        break;
                    }
                }while(1);

                //次に、上に行ける所まで
                do{
                    cout << "dx: " << datum_x << " dy: " << datum_y << endl;

                    r1p = piece[datum_x][datum_y].up;
                    r1x = piece[piece[datum_x][datum_y].ux][piece[datum_x][datum_y].uy].rx;
                    r1y = piece[piece[datum_x][datum_y].ux][piece[datum_x][datum_y].uy].ry;

                    r2p = piece[piece[datum_x][datum_y].rx][piece[datum_x][datum_y].ry].up;
                    r2x = piece[piece[datum_x][datum_y].rx][piece[datum_x][datum_y].ry].ux;
                    r2y = piece[piece[datum_x][datum_y].rx][piece[datum_x][datum_y].ry].uy;

                    if((r1x == r2x) && (r1y == r2y) && (r1p >= 50 || r2p >= 50 )){
                        cout << "up" << endl;
```

```
                        int tmp;
                        tmp = datum_x;
                        datum_x = piece[datum_x][datum_y].ux;
                        datum_y = piece[tmp][datum_y].uy;
                }else{
                        break;
                }
        }while(1);

        //左上に来たため、順番に右に進んでいく
        int xx, yy;
        xx = datum_x;
        yy = datum_y;

        for(int pos_y=0; pos_y<piece_y; pos_y++){
                for(int pos_x=0; pos_x<piece_x; pos_x++){
                        cout << xx << "," << yy << "=" << pos_x << "," << pos_y
                            << endl;
                        piece[xx][yy].re_pos_x = pos_x;
                        piece[xx][yy].re_pos_y = pos_y;

                        int tmp;
                        tmp = xx;
                        xx = piece[xx][yy].rx;
                        yy = piece[tmp][yy].ry;
                }
                xx = piece[datum_x][datum_y].dx;
                yy = piece[datum_x][datum_y].dy;
                datum_x = xx;
                datum_y = yy;
        }
        //glutInitWindowPosition(100, 100);

         pic_data.imageGlutInitWindowSize();
         glutInit(&argc, argv);
         glutInitDisplayMode(GLUT_RGBA);
         glutCreateWindow("create_image");
         glutReshapeFunc(resize);
         init();

         glutDisplayFunc(display);

         glutMainLoop();

        //メモリの解放呼び出されて無いかも。。。
        pic_data.delete_data();
        for(int i=0; i<piece_y; i++){
                for(int j=0; j<piece_x; j++){
                        piece[j][i].delete_data();
                }
        }
        for(int i=0; i<piece_x; i++){
                delete[] piece[i];
        }
        delete[] piece;
        cout << "create delete PIECE memory" << endl;

        for(int i=0; i<piece_x; i++){
                delete[] compare_r[i];
                delete[] compare_l[i];
                delete[] compare_u[i];
                delete[] compare_d[i];
        }
        delete[] compare_r;
        delete[] compare_l;
        delete[] compare_u;
        delete[] compare_d;
        cout << "create delete compare memory" << endl;
}
```

List 8: sort/Pos.cpp

```
#include "Pos.h"
// ポジション

Pos::Pos() {
  x = y = -1;
}

Pos::Pos(int x, int y) {
  this->x = x;
  this->y = y;
}

void Pos::setZero() {
  this->x = this->y = 0;
}
```

List 9: sort/Pos.h

```
#ifndef INCLUDED_POS_H
#define INCLUDED_POS_H

class Pos {
public:
  int x, y;

  Pos();
  Pos(int x, int y);
  void setZero();
};

#endif
```

List 10: sort/PosData.cpp

```
#include "PosData.h"
#include "stdio.h"
#include "stdlib.h"
#include "util.h"

PosData::PosData() {
}

PosData::PosData(int w, int h) {
  int i;

  width = w;
  height = h;
  data = new Pos*[height];
  for(i = 0; i < height; i++) {
    data[i] = new Pos[width];
  }
}

PosData::~PosData() {
  int i;

  for(i = 0; i < height; i++) {
    delete [] data[i];
  }
  delete [] data;
}

void PosData::dispData() {
  int i, j;

  for(i = 0; i < height; i++) {
    for(j = 0; j < width; j++) {
      printf("%X%X ", data[i][j].x, data[i][j].y);
    }
    puts("");
  }
  puts("");
}

int PosData::getHeight() {
  return this->height;
}
int PosData::getWidth() {
  return this->width;
}

int PosData::getX(int ox, int oy) {
  if(!checkInScope(width, height, ox, oy)) myerror(1);
  return data[oy][ox].x;
}

int PosData::getY(int ox, int oy) {
  if(!checkInScope(width, height, ox, oy)) myerror(1);
  return data[oy][ox].y;
}

void PosData::randomizeData() {
  int i, j, x, y;

  for(i = 0; i < height; i++) {
    for(j = 0; j < width; j++) {
      x = rand() % width;
      y = rand() % height;
      swapPos(&data[i][j], &data[y][x]);
    }
  }
}

void PosData::setX(int ox, int oy, int x) {
  if(!checkInScope(width, height, ox, oy)) myerror(1);
  data[oy][ox].x = x;
}

void PosData::setY(int ox, int oy, int y) {
  if(!checkInScope(width, height, ox, oy)) myerror(1);
  data[oy][ox].y = y;
}

void PosData::setData(int ox, int oy, int x, int y) {
  data[oy][ox].x = x;
  data[oy][ox].y = y;
}
```

List 11: sort/PosData.h

```
#ifndef INCLUDED_POSDATA_H
#define INCLUDED_POSDATA_H
#include "Pos.h"

class PosData {
```

```cpp
private:
  Pos **data;
  int width, height;

public:
  PosData();
  PosData(int w, int h);
  ~PosData();
  void dispData();
  int getHeight();
  int getWidth();
  int getX(int ox, int oy);
  int getY(int ox, int oy);
  void randomizeData();
  void setX(int ox, int oy, int x);
  void setY(int ox, int oy, int y);
  void setData(int ox, int oy, int x, int y);
};

#endif
```

List 12: sort/Process5.cpp

```cpp
#include <stdio.h>
#include <stdlib.h>
#include "Process5.h"
#include "util.h"

using namespace std;

Pro5::Process5(): ProcessBase() {
  target.setZero();
  target_data.setZero();
}

Pro5::Process5(int w, int h): ProcessBase(w, h) {
  target.setZero();
  target_data.setZero();
}

void Pro5::dispSorted() {
  list<Pos>::iterator p;

  for(p = sorted.begin(); p != sorted.end(); p++) {
    printf("%d, %d\n", p->x, p->y);
  }
  puts("");
}

void Pro5::dispSortedData() {
  list<Pos>::iterator p;

  int i, j;

  for(i = 0; i < table->getHeight(); i++) {
    for(j = 0; j < table->getWidth(); j++) {
      for(p = sorted.begin(); p != sorted.end(); p++) {
        if(checkPosEqual(j, i, p->x, p->y))
          changeWordColor(RED);
      }
      if(checkPosEqual(j, i, table->getSelected().x, table->getSelected().y))
        changeWordColor(GREEN);
      printf("%X%X ", table->getData(j, i).x, table->getData(j, i).y);
      defaultWordColor();
    }
    puts("");
  }
  puts("");
}

// 安全
int Pro5::isSelectedNextToTarget() {
  Pos s = table->getSelected();

  return isNext(s, target);
}

int Pro5::isSorted(int y) {
  int x;
  Pos data;

  for(x = 0; x < table->getWidth(); x++) {
    data = table->getData(x, y);
    if(data.x != x || data.y != y) return 0;
  }
  return 1;
}

void Pro5::moveSelected(Pos destination) {
  Pos s = table->getSelected();
  list<Pos>::iterator p;

  int directionLR;
  int directionUD;
  int move_dir = 0;
  int move_flag = 0;
  int half;
  int old_direction = -1;

  puts("-------moveSelected-------");

  while(!checkPosEqual(destination, s)) {
    s = table->getSelected();
    //half = s.y / (table->getHeight() / 2);
    half = 1;
    directionLR = getDirectionLR(s.x, destination.x);
    directionUD = getDirectionUD(s.y, destination.y);
    table->dispData();
    if(half == 1) {
      if(directionLR == EQUAL) {
        move_dir = directionUD;
        move_flag = UD;
      } else {
        move_dir = directionLR;
        move_flag = LR;
      }
    } else if(half == 0) {
      if(directionUD == EQUAL) {
        move_dir = directionLR;
        move_flag = LR;
      } else {
        move_dir = directionUD;
        move_flag = UD;
      }
    }
    for(p = sorted.begin(); p != sorted.end(); p++) {
      if(checkPosEqual(surroundings(s, move_dir), *p)) {
        if(move_flag == LR) {
          move_dir = directionUD;
          move_flag = UD;
          break;
        } else if(move_flag == UD) {
          move_dir = directionLR;
          move_flag = LR;
          break;
        }
      }
    }
    for(p = sorted.begin(); p != sorted.end(); p++) {
      if(checkPosEqual(surroundings(s, move_dir), *p)) {
        if(move_flag == LR) {
          move_dir = getReversedDirection(directionLR);
          break;
        } else if(move_flag == UD) {
          move_dir = getReversedDirection(directionUD);
          break;
        }
      }
    }
    for(p = sorted.begin(); p != sorted.end(); p++) {
      if(checkPosEqual(surroundings(s, move_dir), *p)) {
        if(move_flag == LR) {
          move_dir = getReversedDirection(directionUD);
          move_flag = UD;
          break;
        } else if(move_flag == UD) {
          move_dir = getReversedDirection(directionLR);
          move_flag = LR;
          break;
        }
      }
    }
    /*
    if(old_direction == move_dir) {
      move_dir = getReversedDirection(move_dir);
    }*/
    printf("move_dir = %d\n", move_dir);
    table->swapSelected(move_dir);
    table->dispData(target.x, target.y);
    old_direction = getReversedDirection(move_dir);
  }
  puts("========moveSelected end=======");
}

void Pro5::moveSelectedNextTarget() {
  Pos s = table->getSelected();
  list<Pos>::iterator p;

  int directionLR;
  int directionUD;
  int move_dir = 0;
  int move_flag = 0;
  int half = 1;
  int old_direction = -1;

  puts("-------moveSelectedNextTarget-------");
  table->dispData(target.x, target.y);

  while(!isSelectedNextToTarget()) {
    s = table->getSelected();
    directionLR = getDirectionLR(s.x, target.x);
    directionUD = getDirectionUD(s.y, target.y);
    if(half == 1) {
      if(isNextX(target, s)) {
        move_dir = directionUD;
        move_flag = UD;
      } else {
        move_dir = directionLR;
        move_flag = LR;
      }
    } else if(half == 0) {
      if(isNextY(target, s)) {
        move_dir = directionLR;
        move_flag = LR;
      } else {
        move_dir = directionUD;
        move_flag = UD;
      }
    }
    for(p = sorted.begin(); p != sorted.end(); p++) {
      if(checkPosEqual(surroundings(s, move_dir), *p)) {
        if(move_flag == LR) {
          move_dir = directionUD;
          move_flag = UD;
          break;
        } else if(move_flag == UD) {
          move_dir = directionLR;
          move_flag = LR;
          break;
        }
      }
    }
    for(p = sorted.begin(); p != sorted.end(); p++) {
      if(checkPosEqual(surroundings(s, move_dir), *p)) {
        if(move_flag == LR) {
          move_dir = getReversedDirection(directionLR);
          break;
        } else if(move_flag == UD) {
          move_dir = getReversedDirection(directionUD);
          break;
        }
      }
    }
    for(p = sorted.begin(); p != sorted.end(); p++) {
      if(checkPosEqual(surroundings(s, move_dir), *p)) {
        if(move_flag == LR) {
          move_dir = getReversedDirection(directionUD);
          move_flag = UD;
          break;
        } else if(move_flag == UD) {
          move_dir = getReversedDirection(directionLR);
          move_flag = LR;
          break;
        }
      }
    }
    /*
    for(p = sorted.begin(); p != sorted.end(); p++) {
      if(checkPosEqual(surroundings(s, move_dir), *p)) {
        if(move_flag == LR) {
          move_dir = directionUD;
          move_flag = UD;
          break;
        } else if(move_flag == UD) {
          move_dir = directionLR;
          move_flag = LR;
          break;
        }
      }
    }
    if(old_direction == move_dir) {
      move_dir = getReversedDirection(move_dir);
    }
    for(p = sorted.begin(); p != sorted.end(); p++) {
      if(checkPosEqual(surroundings(s, move_dir), *p)) {
        if(move_flag == LR) {
          move_dir = getReversedDirection(directionUD);
          move_flag = UD;
          break;
        } else if(move_flag == UD) {
          move_dir = getReversedDirection(directionLR);
          move_flag = LR;
          break;
        }
      }
    }*/
    table->swapSelected(move_dir);
    table->dispData(target.x, target.y);
    old_direction = getReversedDirection(move_dir);
  }
  puts("=======moveSelectedNextTarget=======");
}

int Pro5::moveTarget(Pos pos) {
  list<Pos>::iterator p;

  int directionLR;
  int directionUD;
  int move_dir = 0;
  int move_flag = 0;
  //int half = target.y / (table->getHeight() / 2);
  int old_direction = -1;
  int old_flag = -1;

  puts("-------moveTarget-------");

  while(!checkPosEqual(target, pos)) {
    target = table->findData(target_data);
    directionLR = getDirectionLR(target.x, pos.x);
    directionUD = getDirectionUD(target.y, pos.y);
    table->dispData();
    if(old_flag == LR) {
      if(directionUD == EQUAL) {
        move_dir = directionLR;
        move_flag = LR;
      } else {
        move_dir = directionUD;
        move_flag = UD;
      }
    } else {
      if(directionLR == EQUAL) {
        move_dir = directionUD;
        move_flag = UD;
      } else {
        move_dir = directionLR;
        move_flag = LR;
      }
    }
    for(p = sorted.begin(); p != sorted.end(); p++) {
      if(checkPosEqual(surroundings(target, move_dir), *p)) {
        if(move_flag == LR) {
          move_dir = directionUD;
          move_flag = UD;
          break;
        } else if(move_flag == UD) {
          move_dir = directionLR;
          move_flag = LR;
          break;
        }
      }
    }
    if(old_direction == move_dir) {
      move_dir = getReversedDirection(move_dir);
    }
```

```cpp
    for(p = sorted.begin(); p != sorted.end(); p++) {
      if(checkPosEqual(surroundings(target, move_dir), *p)) {
        if(move_flag == LR) {
          move_dir = getReversedDirection(directionUD);
          move_flag = UD;
          break;
        } else if(move_flag == UD) {
          move_dir = getReversedDirection(directionLR);
          move_flag = LR;
          break;
        }
      }
    }
    printf("directionaaaaa%d, %d, %d\n", directionLR, directionUD, move_dir);
    moveSelectedNextTarget();
    rotateSelected(move_dir);
    table->swapSelected(getReversedDirection(move_dir));
    table->dispData(target.x, target.y);
    old_direction = getReversedDirection(move_dir);
    old_flag = move_flag;
  }
  puts("=======moveTarget end========");
  return 0;
}

void Pro5::rotateSelected(int direction) {
  // directionは動かす方向
  // targetから見たselectedの方向
  int dir_selected = getDirection(target, table->getSelected());
  int move_distance = abs(direction - dir_selected);
  // 1か-1
  // 間違ってる
  //------------------------------直すところ
  int move_direction = (dir_selected > direction) ? -1 : 1;

  puts("-------rotateSelected-------");
  printf("direction = %d\n", direction);
  printf("target = %d, %d\n", target.x, target.y);
  if(direction == EQUAL) return;
  if(checkPosEqual(table->getSelected(), surroundings(target, direction)))  {
    puts("selected dont need to rotate");
    return;
  }
  if(!checkInScope(table->getWidth(), table->getHeight(), surroundings(target, direction).x, surroundings(target, direction).y)) {
    puts("cant rotate(move target)...");
    return;
  }

  if(move_distance > DIRECTION_NUM / 2) {
    puts("rotate distance is too long");
    move_distance = DIRECTION_NUM - move_distance;
    move_direction *= -1;
  }

  Pos dummy_target = target;
  int i;
  int reverse_flag = 0;
  list<Pos>::iterator p;

  for(i = dir_selected; i != direction; i = (i + move_direction + DIRECTION_NUM) % DIRECTION_NUM) {
    if(!checkInScope(table->getWidth(), table->getHeight(), surroundings(dummy_target, i).x, surroundings(dummy_target, i).y)) reverse_flag = 1;
    for(p = sorted.begin(); p != sorted.end(); p++) {
      if(checkPosEqual(surroundings(dummy_target, i), *p)) {
        reverse_flag = 1;
        break;
      }
    }
    if(reverse_flag) {
      move_distance = DIRECTION_NUM - move_distance;
      move_direction *= -1;
      break;
    }
  }

  int move_dir;
  int j = dir_selected;
  printf("move_direction = %d\n", move_direction);
  printf("move_distance = %d\n", move_distance);
  for(i = 0; i < move_distance; i++) {
    // 目的地についたら止めたほうがいいかも
    j = (j + move_direction + DIRECTION_NUM) % DIRECTION_NUM;
    move_dir = getDirection(table->getSelected(), surroundings(target, j));
    printf("j = %d\n", j);
    printf("move_dir = %d\n", move_dir);
    table->swapSelected(move_dir);
    table->dispData(target.x, target.y);
  }
  puts("======rotate end========");
}

string Pro5::sort() {
  table->dispData();
  // 一番右下になるデータを選択
  table->findAndSelectData(table->getWidth()-1, table->getHeight()-1);
  table->dispData();

  sortUp();
  dispSortedData();
  sortDown();
  dispSortedData();
  table->dispData();
  table->dispCost();

  return table->getStringSortData();
}

void Pro5::sortDown() {
  int i;

  if(isSorted(table->getHeight()-2) && isSorted(table->getHeight()-1)) return;
  for(i = 0; i < table->getWidth()-2; i++) {
    Pos dummy = table->findData(Pos(i, table->getHeight()-2));
    if(dummy.x >= i && dummy.x < i+2) {
      target_data = Pos(i, table->getHeight()-2);
      target = table->findData(target_data);
      moveTarget(Pos(i+2, dummy.y));
    }
    target_data = Pos(i, table->getHeight()-1);
    target = table->findData(target_data);
    moveTarget(Pos(i, table->getHeight()-2));
    sorted.push_back(Pos(i, table->getHeight()-2));
    target_data = Pos(i, table->getHeight()-2);
    target = table->findData(target_data);
    moveTarget(Pos(i+1, table->getHeight()-2));
    sorted.push_back(Pos(i+1, table->getHeight()-2));

    moveSelected(Pos(i, table->getHeight()-1));
    table->swapSelected(UP);
    table->swapSelected(RIGHT);
    sorted.pop_back();
    sorted.pop_back();
    sorted.push_back(Pos(i, table->getHeight()-1));
    sorted.push_back(Pos(i, table->getHeight()-2));
    table->dispData();
  }
  target_data = Pos(table->getWidth()-2, table->getHeight()-2);
  target = table->findData(target_data);
  moveTarget(target_data);
  sorted.push_back(Pos(table->getWidth()-2, table->getHeight()-2));
  table->dispData();

  target_data = Pos(table->getWidth()-2, table->getHeight()-1);
  target = table->findData(target_data);
  table->dispData(target.x, target.y);
  if(checkPosEqual(target, Pos(table->getWidth()-1, table->getHeight()-2))) {
    table->selectData(target.x, target.y);
    moveSelected(target_data);
  }
  target_data = Pos(table->getWidth()-1, table->getHeight()-2);
  target = table->findData(target_data);
  table->dispData(target.x, target.y);
  if(checkPosEqual(target, Pos(table->getWidth()-2, table->getHeight()-1))) {
    table->selectData(target.x, target.y);
    moveSelected(target_data);
  }
  target_data = Pos(table->getWidth()-1, table->getHeight()-1);
  target = table->findData(target_data);
  table->selectData(target.x, target.y);
  moveSelected(target_data);
}

void Pro5::sortUp() {
  int i, j;
  // 上半分
  for(i = 0; i < table->getHeight()-2; i++) {
    // 端以外の
    if(isSorted(i)) {
      puts("continue1");
      for(j = 0; j < table->getWidth(); j++) {
        sorted.push_back(Pos(j, i));
      }
      continue;
    }
    for(j = 0; j < table->getWidth()-2; j++) {
      target_data = Pos(j, i);
      target = table->findData(target_data);
      table->dispData(target.x, target.y);
      // 目的地へ移動
      moveTarget(target_data);
      target = table->findData(target_data);
      sorted.push_back(target);
    }
    // もし、1列全部揃ってたらcontinue
    if(isSorted(i)) {
      puts("continue2");
      sorted.push_back(Pos(table->getWidth()-2, i));
      sorted.push_back(Pos(table->getWidth()-1, i));
      continue;
    }
    Pos dummy = table->findData(Pos(table->getWidth()-2, i));
    if(dummy.x >= table->getWidth()-2 && dummy.x < table->getWidth() && dummy.y
        >= i && dummy.y < i+2) {
      target_data = Pos(table->getWidth()-2, i);
      target = table->findData(target_data);
      moveTarget(Pos(table->getWidth()-2, i+2));
    }
    // 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
    if(!checkPosEqual(table->getData(table->getWidth()-2, i), Pos(table->getWidth
      ()-1, i)) || !checkPosEqual(table->getData(table->getWidth()-2, i+1), Pos(
      table->getWidth()-2, i))) {
      target_data = Pos(table->getWidth()-1, i);
      target = table->findData(target_data);
      moveTarget(Pos(table->getWidth()-2, i));
      sorted.push_back(Pos(table->getWidth()-2, i));
      target_data = Pos(table->getWidth()-2, i);
      target = table->findData(target_data);
      moveTarget(Pos(table->getWidth()-2, i+1));
      sorted.push_back(Pos(table->getWidth()-2, i+1));
    } else {
    }
    moveSelected(Pos(table->getWidth()-1, i));
    table->swapSelected(LEFT);
    table->swapSelected(DOWN);
    sorted.pop_back();
    sorted.pop_back();
    sorted.push_back(Pos(table->getWidth()-2, i));
    sorted.push_back(Pos(table->getWidth()-1, i));
    table->dispData();
  }
}
```

List 13: sort/Process5.h

```
#ifndef INCLUDED_PROCESS5_H
#define INCLUDED_PROCESS5_H
#define LR 0
#define UD 1
#include "ProcessBase.h"
#include <list>

using namespace std;

typedef class Process5 : public ProcessBase {
private:
  Pos target;
  Pos target_data;
  list<Pos> sorted;

private:
  Process5();
  void dispSorted();
  void dispSortedData();
  int isSelectedNextToTarget();
  int isSorted(int y);
  void moveSelected(Pos destination);// なにか間違っているかも
  void moveSelectedNextTarget();
  int moveTarget(Pos pos);
  void rotateSelected(int direction);
  void sortDown();
  void sortUp();
public:
  Process5(int w, int h);
  string sort();
} Pro5;

// 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.

#endif
```

List 14: sort/ProcessBase.cpp

```
#include <stdio.h>
#include "ProcessBase.h"
```

```

ProBase::ProcessBase() {
}

ProBase::ProcessBase(int width, int height) {
  table = new Dataset(width, height);
  table->randomizeData();
}

ProBase::~ProcessBase() {
  delete table;
}

void ProBase::importData(PosData &data) {
  table->importData(data);
}
```

List 15: sort/ProcessBase.h

```
#ifndef INCLUDED_PROCESSBASE_H
#define INCLUDED_PROCESSBASE_H
#include "dataset.h"

typedef class ProcessBase {
protected:
  Dataset *table;

protected:
  ProcessBase();
public:
  ProcessBase(int width, int height);
  ~ProcessBase();
  void importData(PosData &data);
} ProBase;

#endif
```

List 16: sort/util.cpp

```
#include <stdio.h>
#include <iostream>
#include <string>
#include <stdlib.h>
#include "util.h"

using namespace std;

int checkPosEqual(int x1, int y1, int x2, int y2) {
  return ((x1 == x2) && (y1 == y2));
}

int checkPosEqual(Pos p1, Pos p2) {
  return checkPosEqual(p1.x, p1.y, p2.x, p2.y);
}

int getDirection(Pos source, Pos destination) {
  if(source.x == destination.x && source.y == destination.y) {
    return EQUAL;
  } else if(source.x == destination.x && source.y > destination.y) {
    return UP;
  } else if(source.x < destination.x && source.y > destination.y) {
    return UPPERRIGHT;
  } else if(source.x < destination.x && source.y == destination.y) {
    return RIGHT;
  } else if(source.x < destination.x && source.y < destination.y) {
    return LOWERRIGHT;
  } else if(source.x == destination.x && source.y < destination.y) {
    return DOWN;
  } else if(source.x > destination.x && source.y < destination.y) {
    return LOWERLEFT;
  } else if(source.x > destination.x && source.y == destination.y) {
    return LEFT;
  } else if(source.x > destination.x && source.y > destination.y) {
    return UPPERLEFT;
  }
  return EQUAL;
}

int getReversedDirection(int direction) {
  return (direction + DIRECTION_NUM / 2) % DIRECTION_NUM;
}

int getDirectionLR(int ox, int x) {
  if(ox == x) return EQUAL;
  return (ox < x) ? RIGHT : LEFT;
}

int getDirectionUD(int oy, int y) {
  if(oy == y) return EQUAL;
  return (oy < y) ? DOWN : UP;
}

string getDirectionChar(int direction) {
  switch(direction) {
  case UP:
    return "U";
    break;
  case RIGHT:
    return "R";
    break;
  case DOWN:
    return "D";
    break;
  case LEFT:
    return "L";
    break;
  }
  return "";
}

int isConnected(Pos p1, Pos p2) {
  return ((p1.x-1 == p2.x && p1.y == p2.y) || (p1.x+1 == p2.x && p1.y == p2.y) ||
       (p1.x == p2.x && p1.y-1 == p2.y) || (p1.x == p2.x && p1.y+1 == p2.y));
}

int isNext(Pos p1, Pos p2) {
  return ((p1.x+1 == p2.x || p1.x-1 == p2.x || p1.x == p2.x) && (p1.y-1 == p2.y
      || p1.y+1 == p2.y || p1.y == p2.y));
}

int isNextX(Pos p1, Pos p2) {
  return (p1.x+1 == p2.x || p1.x-1 == p2.x || p1.x == p2.x);
}

int isNextY(Pos p1, Pos p2) {
  return (p1.y-1 == p2.y || p1.y+1 == p2.y || p1.y == p2.y);
}

void myerror(int error_code) {
  puts("MYERROR");
  exit(1);
}
```

```

// directionの方向にxとyを移動する
void surroundings(int *x, int *y, int direction) {
  switch(direction) {
  case UP:
    (*y)--;
    break;
  case RIGHT:
    (*x)++;
    break;
  case DOWN:
    (*y)++;
    break;
  case LEFT:
    (*x)--;
    break;
  case UPPERRIGHT:
    (*x)++;
    (*y)--;
    break;
  case LOWERRIGHT:
    (*x)++;
    (*y)++;
    break;
  case LOWERLEFT:
    (*x)--;
    (*y)++;
    break;
  case UPPERLEFT:
    (*x)--;
    (*y)--;
    break;
  }
}

Pos surroundings(Pos Data, int direction) {
  switch(direction) {
  case UP:
    Data.y--;
    break;
  case RIGHT:
    Data.x++;
    break;
  case DOWN:
    Data.y++;
    break;
  case LEFT:
    Data.x--;
    break;
  case UPPERRIGHT:
    Data.x++;
    Data.y--;
    break;
  case LOWERRIGHT:
    Data.x++;
    Data.y++;
    break;
  case LOWERLEFT:
    Data.x--;
    Data.y++;
    break;
  case UPPERLEFT:
    Data.x--;
    Data.y--;
    break;
  }
  return Data;
}

// num1とnum2の値を交換
void swapNum(int *num1, int *num2) {
  int dummy = *num1;
  *num1 = *num2;
  *num2 = dummy;
}

// p1とp2の値を交換する
void swapPos(Pos *p1, Pos *p2) {
  Pos dummy = *p1;
  *p1 = *p2;
  *p2 = dummy;
}
```

List 17: sort/util.h

```
#ifndef INCLUDED_UTIL_H
#define INCLUDED_UTIL_H
#include "Pos.h"
#include <stdio.h>
#include <iostream>
#include <string>

using namespace std;

#define EQUAL -1
#define UP 0
#define UPPERRIGHT 1
#define RIGHT 2
#define LOWERRIGHT 3
#define DOWN 4
#define LOWERLEFT 5
#define LEFT 6
#define UPPERLEFT 7
#define DIRECTION_NUM 8
#define BLACK 0
#define RED 1
#define GREEN 2
#define YELLOW 3
#define BLUE 4
#define MAGENTA 5
#define CYAN 6
#define WHITE 7

#define changeWordColor(cc) printf("\033[3%dm", cc)
#define changeBackColor(cc) printf("\033[4%dm", cc)
#define checkInScope(width, height, x, y) (x>=0 && x<width && y >=0 && y<height)
#define convertHex(x, y) x+y*0x10
#define convertX(num) num%0x10
#define convertY(num) num/0x10
#define defaultWordColor() printf("\033[39m")
#define defaultBackColor() printf("\033[49m")

extern int checkPosEqual(int x1, int y1, int x2, int y2);
extern int checkPosEqual(Pos p1, Pos p2);
extern int getDirection(Pos source, Pos destination);
extern int getDirectionLR(int ox, int x);
extern int getDirectionUD(int oy, int y);
extern string getDirectionChar(int direction);
extern int getReversedDirection(int direction);
extern int isConnected(Pos p1, Pos p2);
extern int isNext(Pos p1, Pos p2);
extern int isNextX(Pos p1, Pos p2);
extern int isNextY(Pos p1, Pos p2);
```

```c
extern void myerror(int error_code);
extern void swapNum(int *num1, int *num2); // num1とnum2を交換
extern void surroundings(int *x, int *y, int direction);
extern Pos surroundings(Pos Data, int direction);
extern void swapPos(Pos *p1, Pos *p2);

#endif
```