struct FegyverLista

{

std::string nev;

std::string tulajdonos;

FegyverLista\* next;

};

struct TulajdonosLista

{

std::string nev;

TulajdonosLista\* next;

};

FegyverLista\* beszurFegyver(FegyverLista\* lista, unsigned int i\_pos, const std::string& line)

{

FegyverLista\* ujelem = new FegyverLista();

ujelem->nev = line.substr(0, i\_pos);

ujelem->tulajdonos = line.substr(i\_pos+1, line.length());

ujelem->next = NULL;

if (lista == NULL)

{

lista = ujelem;

}

else

{

lista->next = ujelem;

lista = lista->next;

}

return lista;

}

FegyverLista\* beolvasasFegyverek(const std::string filename)

{

std::ifstream f(filename.c\_str());

std::string line;

FegyverLista\* lista = NULL;

FegyverLista\* cursor = NULL;

do

{

std::getline(f, line);

for (unsigned int i\_pos = 0; i\_pos < line.length(); i\_pos++)

{

if (line[i\_pos] == ',')

{

cursor = beszurFegyver(cursor, i\_pos, line);

if (lista == NULL)

{

lista = cursor;

}

break;

}

}

} while (!f.eof());

return lista;

}

TulajdonosLista\* szures(FegyverLista\* lista, const std::string fegyvernev)

{

TulajdonosLista\* tulajdonos\_start = NULL;

TulajdonosLista\* tulajdonos\_cursor = NULL;

// Fegyverek kezelese

FegyverLista\* fegyver\_cursor = lista;

while (fegyver\_cursor != NULL)

{

if (fegyver\_cursor->nev.compare(fegyvernev) == 0)

{

TulajdonosLista\* ujelem = new TulajdonosLista();

ujelem->nev = fegyver\_cursor->tulajdonos;

ujelem->next = NULL;

if (tulajdonos\_start==NULL)

{

tulajdonos\_start = ujelem;

tulajdonos\_cursor = tulajdonos\_start;

}

else

{

tulajdonos\_cursor->next = ujelem;

tulajdonos\_cursor = tulajdonos\_cursor->next;

}

}

fegyver\_cursor = fegyver\_cursor->next;

}

return tulajdonos\_start;

}

struct Robot

{

std::string name;

std::string country;

double height;

unsigned int year;

};

struct RobotLink

{

Robot\* robot;

RobotLink\* next;

};

RobotLink\* robotOrszagonkent(const std::string country\_name, Robot\* robots, unsigned int n)

{

RobotLink\* first = NULL;

RobotLink\* cursor = NULL;

for (unsigned int i = 0; i < n; i++)

{

if (robots[i].country.compare(country\_name) == 0)

{

RobotLink\* ujelem = new RobotLink();

ujelem->robot = &robots[i];

ujelem->next = NULL;

if (first == NULL)

{

first = ujelem;

cursor = first;

}

else

{

cursor->next = ujelem;

}

}

}

return first;

}

void printRobots(RobotLink\* robots)

{

RobotLink\* cursor = robots;

while (cursor != NULL)

{

std::cout << cursor->robot->name << ' ' << cursor->robot->country << ' ' << cursor->robot->height << ' ' << cursor->robot->year << '\n';

cursor = cursor->next;

}

}

void robotStatistics(RobotLink\* robots)

{

RobotLink\* cursor = robots;

Robot\* magasabb = NULL;

double max\_height = 0.0;

while (cursor != NULL)

{

if (cursor->robot->height > max\_height)

{

magasabb = cursor->robot;

max\_height = magasabb->height;

}

cursor = cursor->next;

}

if (magasabb != NULL)

{

std::cout << "LEGMAGASABB " << magasabb->country << '\n';

std::cout << magasabb->name << ' ' << magasabb->height << ' ' << magasabb->year << '\n';

}

}

struct Urhajos

{

std::string nev;

double magassag;

unsigned int bevetesek;

};

Urhajos\* keresLegalacsonyabbUrhajos(Urhajos\* urhajosok, unsigned int n)

{

if (n == 0)

{

return NULL;

}

Urhajos\* legkisebbUrhajos = &urhajosok[0];

for (unsigned int i = 0; i < n; i++)

{

if (urhajosok[i].magassag < legkisebbUrhajos->magassag)

{

legkisebbUrhajos = &urhajosok[i];

}

}

return legkisebbUrhajos;

}

Urhajos\* keresLegtobbBevetes(Urhajos\* urhajosok, unsigned int n)

{

if (n == 0)

{

return NULL;

}

Urhajos\* legtobbBevetesUrhajos = &urhajosok[0];

for (unsigned int i = 0; i < n; i++)

{

if ((urhajosok[i].bevetesek == legtobbBevetesUrhajos->bevetesek &&

urhajosok[i].magassag > legtobbBevetesUrhajos->magassag)||

urhajosok[i].bevetesek > legtobbBevetesUrhajos->bevetesek)

{

legtobbBevetesUrhajos = &urhajosok[i];

}

}

return legtobbBevetesUrhajos;

}

Question author's solution:

enum LevegoMinoseg

{

UNDEF,

JO,

KOZEPES,

KELLEMETLEN,

EGESZSEGTELEN,

VESZELYES

};

void printKategoria(const LevegoMinoseg& cat)

{

switch (cat)

{

case JO:

{

std::cout << "JO\n";

break;

}

case KOZEPES:

{

std::cout << "KOZEPES\n";

break;

}

case KELLEMETLEN:

{

std::cout << "KELLEMETLEN\n";

break;

}

case EGESZSEGTELEN:

{

std::cout << "EGESZSEGTELEN\n";

break;

}

case VESZELYES:

{

std::cout << "VESZELYES\n";

break;

}

default:

{

std::cout << "UNDEF\n";

break;

}

}

}

LevegoMinoseg vizsgal(int val)

{

if (val < 0)

{

return UNDEF;

}

else if (val <= 50)

{

return JO;

}

else if (val <= 100)

{

return KOZEPES;

}

else if (val <= 150)

{

return KELLEMETLEN;

}

else if (val <= 200)

{

return EGESZSEGTELEN;

}

else

{

return VESZELYES;

}

}

void beolvasNo2(const std::string filename)

{

std::ifstream f(filename.c\_str());

std::string line;

int t = 0.0;

LevegoMinoseg cat = UNDEF;

double sum = 0.0;

unsigned int cnt = 0;

while (!f.eof())

{

std::getline(f, line);

t = std::atoi(line.c\_str());

if (t >= 0)

{

LevegoMinoseg newcat = vizsgal(t);

if (cat != newcat)

{

printKategoria(newcat);

cat = newcat;

}

sum += t;

cnt++;

}

}

if (cnt != 0)

{

double no2atlag = sum / cnt;

std::cout << "NO2 koncentráció átlaga: " << no2atlag << '\n';

printKategoria(vizsgal(no2atlag));

}

}

Question author's solution:

double\* createRotation(const double alpha)

{

double \*rot = new double[4];

rot[0] = cos(alpha);

rot[1] = -sin(alpha);

rot[2] = sin(alpha);

rot[3] = cos(alpha);

return rot;

}

double\* createTranslation(const double x, const double y)

{

double\* tr = new double[2];

tr[0] = x;

tr[1] = y;

return tr;

}

double\*\* convertTo2DHomogeneousTransformation(double\* r, double\* t)

{

double\*\* res = new double\*[3];

for (unsigned int i = 0; i < 3; i++)

{

res[i] = new double[3];

for (unsigned int j = 0; j < 3; j++)

res[i][j] = 0.0;

}

for (unsigned int i = 0; i < 2; i++)

{

for (unsigned int j = 0; j < 2; j++)

{

res[i][j] = r[j + 2 \* i];

}

}

for (unsigned int i = 0; i < 3; i++)

{

res[i][2] = t[i];

}

res[2][2] = 1;

return res;

}

void printHomogeneousTransformation(double\*\* hom)

{

std::cout << std::setprecision(3);

std::cout << std::fixed;

std::cout << '['<< '\n';

for (int i = 0; i < 3; i++)

{

std::cout << '[';

for (int j = 0; j < 3; j++)

{

std::cout << hom[i][j] << ',';

}

std::cout << ']' << '\n';

}

std::cout << ']' << '\n';

}