

Készítsen egy programot, ami egy fájból beolvasson NO<sub>2</sub> értékeket! A fájlban minden új sorban egy érték szerepel! A beolvasott adatokon a következő tartományokat vizsgálja:

- A negatív értékekkel ne foglalkozzon.

- 0-50: JO

- 51-100: KOZEPESE

- 101-150: KELLEMETLEN

- 151-200: EGESZSEGTELEN

- 201 <: VESZELYES

Feladata a következő függvények megvalósítása:

- **void beolvasNo2(const std::string filename):** egy fájl helyről beolvassa soronként a fájl tartalmát. Csak az állapotátmenetekkor írja ki a standard outputra a vizsgált értéket: vagyis ha egy új beolvasott érték az előzőhöz képest más kategóriába tartozik! Végül írja ki az adatsor által jegyzett átlagos levegőminőséget!

- **LevegoMinoseg vizsgalint val):** a fenti osztályozás alapján döntse el, hogy a bemeneti integer érték hova tartozik, térjen vissza a minőséget jelző enummal!

Összesítse, hogy a fájlban milyen volt átlagosan a levegő! Ha nem volt értelmes mérhető érték, ne írjon ki semmit!

Az adatok publikus UCI adathalmazból származnak.

**TIPP:** segítségként megadtunk egy enumerációt a kategorizáláshoz és a kategóriák kiíratását standard outputra.

For example:

Test	Result
beolvasNo2("no2_test.txt");	KELLEMETLEN EGESZSEGTELEN KELLEMETLEN NO2 koncentráció átlaga: 136.5 KELLEMETLEN

## Question author's solution:

```
enum LevegoMinoseg
{
    UNDEF,
    JO,
    KOZEPESE,
    KELLEMETLEN,
    EGESZSEGTELEN,
    VESZELYES
};

void printKategoria(const LevegoMinoseg& cat)
{
    switch (cat)
    {
        {
        case JO:
        {
            std::cout << "JO\n";
            break;
        }
        case KOZEPESE:
        {
            std::cout << "KOZEPESE\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;
        }
    }
}
```

```

LevegőMinoseg vizsgal(int val)
{
    if (val < 0)
    {
        return UNDEF;
    }
    else if (val <= 50)
    {
        return JO;
    }
    else if (val <= 100)
    {
        return KOZEPEK;
    }
    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;
    LevegőMinoseg cat = UNDEF;
    double sum = 0.0;
    unsigned int cnt = 0;
    while (!f.eof())
    {
        std::getline(f, line);
        t = std::atof(line.c_str());
        if (t >= 0)
        {
            LevegőMinoseg 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));
    }
}

```

A következő program humancid robotokat tárol egy struktúrártömbben. A struktúra a következőképpen adott:

```
struct Robot
{
    std::string name;
    std::string country;
    double height;
    unsigned int year;
};
```

Készítsen függvényeket:

- **RobotLink\* robotOrszagonkent(const std::string country\_name, Robot\* robots, unsigned int n):** Lekérdezi, egy adott országban található robotokat, bemenetként várja a robotok tömbjét, és a tömb méretét. Térjen vissza egy láncolt listával!
- Írja meg a függvényt, ami kiírja az eredménylistát!
- **void robotStatistics(Robot\* robots):** Az eredménylistát felhasználva írja ki, a legmagasabb humancid robotot, a példában látható módon!

For example:

Test	Result
<pre>Robot robots[] = {     {"Szegedi robotember", "HUH", 100.0, 1962},     {"ICUB", "ITA", 105.0, 2009},     {"HUBO", "KOR", 125.0, 2005},     {"ASIMO", "JAP", 120, 2000},     {"Mao", "PRK", 57.4, 2013},     {"NEECH", "SPA", 165, 2013},     {"Fedor", "RUS", 180, 2019},     {"Atlas", "USA", 175, 2013},     {"Murata", "JAP", 50, 2005} }; unsigned int n = 9; RobotLink* Japan_robotok = robotOrszagonkent("JAP", robots, n); printRobots(Japan_robotok); robotStatistics(Japan_robotok); freeRobotList(Japan_robotok);</pre>	<pre>ASIMO JAP 120 2000 Murata JAP 50 2005 LEGNAGASABB JAP ASIMO 120 2000</pre>

Question author's solution:

```
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 << "LEGNAGASABB " << magasabb->country << '\n';
        std::cout << magasabb->name << ' ' << magasabb->height << ' ' << magasabb->year << '\n';
    }
}
```

A feladat, hogy egy fájlból számúrájok által használt fegyverek csoportja kerül beolvasásra. A fájlban az egyes fegyverekhez a tulajdonos is fel van tüntetve, vagyis egy sor a fájlban (az egyes értékek vesszővel vannak elválasztva):

```
Fegyvernév,Tulajdonos
naginata,Tokugawa Ieyasu
...
```

Ijjon függvényeket, ami beolvassa a fegyvereket egy láncolt listába:

- **FegyverLista\* beolvasasFegyverek(std::string filename):** fájl megnyitása, láncolt lista létrehozása és feltöltése

- **FegyverLista\* beszurFegyver(FegyverLista\* lista, unsigned int i\_pos, std::string& line):** egy láncélem beszúrása egy beolvasott sor alapján. Argumentumban várja a lista aktuális elemét, a vessző pozícióját és a beolvasott sort. Térjen vissza a beszúrt elemmel

Ezután írjon egy függvényt, ami egy másik láncolt listában összegyűjti azokat az embereket, akiknek a függvény argumentumában megadott fegyver van (feltételezheti, hogy mindenkinek egy fegyvere van):

- **TulajdonosLista\* szures(FegyverLista\* lista, const std::string fegyvernev):** visszatér egy másik típusú láncolt listával, amiben csak a **fegyvernev** argumentumban szereplő fegyverrel rendelkező emberek vannak. Bementként egy fegyverlistát vár.

A lista struktúra típusok előre megadottak.

TIPP: beolvasás során először keresse meg a vessző helyét egy beolvasott sorban! Ezután használja a substr függvényt!

For example:

Test	Result
FegyverLista* lista = beolvasasFegyverek("fegyverek.txt"); printfFegyverLista(lista); TulajdonosLista* tul = szures(lista, "katana"); std::cout << "Fegyver: katana\n"; printTulajdonosok(tul); freeFegyverLista(lista); freeTulajdonosLista(tul);	katana Oda Nobunaga\r naginata Tokugawa Ieyasu\r kabutowari Fuma Kotaro\r katana Toyotomi Hideyoshi\r katana Miyamoto Musashi\r naginata Takeda Shingen\r katana Miyohime\r kabutowari Ishikawa Goemon\r wakizashi Shimada Genji Fegyver: katana Oda Nobunaga\r Toyotomi Hideyoshi\r Miyamoto Musashi\r Miyohime\r

## Question author's solution:

```
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;
}

```

Készítsen egy **Urhajos** struktúrát, amiben tárolja a:

- Az űrhajós nevét (**nev** mező).
- Az űrhajós magasságát méterben (**magassag** mező).
- A bevetések számát, amin az űrhajós részt vett (**bevetesek** mező).

Ezután keresse meg egy struktúratömbben a legalacsonyabb űrhajóst, és azt, aki a legtöbb bevetésen vett részt:

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

- **Urhajos\* keresLegtobbBevetes(Urhajos\* urhajosok, int n):** ha kettő űrhajós is ugyanannyi bevetésen vett részt, térjen vissza a magasabb űrhajóssal!

Az *n* paraméter a tömb elemszáma. Kezelje azokat az eseteket is (NULL visszatérési értékkel), amikor üres a bemeneti tömb!

For example:

Test	Result
<pre>Urhajos urhajosok[] = {     {"Buzz Aldrin", 1.78, 4},     {"Neil Armstrong", 1.8, 1},     {"James Irwin", 1.73, 4},     {"John Young", 1.83, 3},     {"Michael Collins", 1.83, 2} }; unsigned int n = 5; Urhajos* urh = keresLegalacsonyabbUrhajos(urhajosok, n); std::cout &lt;&lt; "Legkisebb urhajos: " &lt;&lt; urh-&gt;nev &lt;&lt; ", magassaga " &lt;&lt; urh-&gt;magassag &lt;&lt; " m, bevetesei: " &lt;&lt; urh-&gt;bevetesek &lt;&lt; '\n'; Urhajos* urh2 = keresLegtobbBevetes(urhajosok, n); std::cout &lt;&lt; "Legtobb bevetesen reszt vett urhajos: " &lt;&lt; urh2-&gt;nev &lt;&lt; ", magassaga " &lt;&lt; urh2-&gt;magassag &lt;&lt; " m, bevetesei: " &lt;&lt; urh2-&gt;bevetesek &lt;&lt; '\n';</pre>	<pre>Legkisebb urhajos: James Irwin, magassaga 1.73 m, bevetesei: 4 Legtobb bevetesen reszt vett urhajos: Buzz Aldrin, magassaga 1.78 m, bevetesei: 4</pre>


## Question author's solution:

```
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;
}
```

Készítse el a következő függvényeket, amik segítségével 2D homogén transzformáció hozható létre!

 2D Homogeneous Transformation

A vektorokat és mátrixokat dinamikusan hozza létre, majd szolgáltatja visszatérési értékként! A függvények prototípusai a következők:

- **double\* createRotation(const double alpha):** a 2x2-es forgatásmátrix elemeinek elhelyezése egy 4 elemű vektorban. A mátrix elemeit a következő szabály szerint helyezze el:

[ cos(alpha), -sin(alpha), sin(alpha), cos(alpha)]

- **double\* createTranslation(const double x, const double y):** eltolás vektor készítése a következő szabály szerint:

[x, y]

- **double\*\* convertTo2DHomogeneousTransformation(double\* r, double\* t):** forgásmátrix és eltolásvektor összefűzése a következő szabály szerint:

[R[0], R[1], t[0]]

[R[2], R[3], t[1]]

[0, 0, 1]

- **void printHomogeneousTransformation(double\*\* hom):** az eredményként kapott homogén transzformáció kiírata a közzétett példa formátumának megfelelően. Amennyiben az egyik elem abszolútértéke 0.001-nél kisebb, írjon ki 0-át!

**TIPP:** használja az iomanip setprecision és fixed függvényét, például:

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

```
std::cout << std::fixed;
```

A memóriaterület felszabadítását a tesztesetek elvégzik.

For example:

Test	Result
<pre>double* tr = createTranslation(0.0, 0.0); double *rot = createRotation(0.0); double **hom = convertTo2DHomogeneousTransformation(rot, tr); delete tr; delete rot; printHomogeneousTransformation(hom); freeHomogeneousTransformation(hom);</pre>	<pre>[   [1.000,-0.000,0.000,]   [0.000,1.000,0.000,]   [0.000,0.000,1.000,] ]</pre>

## 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';
}
```