Írjon egy C++ programot, amely beolvas 2 nemnegatív egész számot, amely megadja az ábécé 2 betűjének sorszámát. A program írja ki az ábécé betűit az első sorszámtól a második sorszámmal bezárólag! A betűket nagybetűvel, elválasztójel nélkül felsorolva írja ki!

Figyeljen rá, hogy ha az első sorszám nagyobb, akkor fordított ábécé-sorrendben kell kiírni a betűket!

A sorszámok mindig 1 és 26 közöttiek, ezt nem kell ellenőrizni a megoldás során.

For example:

Input	Result
2 4	BCD
5 1	EDCBA

```
#include <cctype>
#include <iostream>
#include <string>

using namespace std;

int main() {
    int a, b, step = 1;
    cin >> a >> b;
    if (a > b) step = -1;
    for (int i = a - 1; i != b - 1 + step; i += step) {
        cout << char('A' + i);
    }
    cout << endl;
    return 0;
}
```

Írjon egy C++ programot, amely egy IP-címekből álló listából kiválogatja azokat, amelyek legalább kétszer ismétlődnek!

A standard bemenetről először olvasson be egy számot, amely megadja, hány IP-cím fog következni. Majd olvassa be az IP-címeket. Ezután a kimeneten sortöréssel elválasztva sorolja fel azokat az IP-címeket, amelyek legalább kétszer szerepeltek a bemeneten.

A kiírásban már minden ismétlődő IP-cím csak egyszer szerepeljen, és **olyan sorrendben legyenek, amilyen sorrendben először előfordultak a bemeneten!**

For example:

Input	Result
8 192.168.0.135 192.168.0.149 192.168.0.111 192.168.0.149 192.168.0.34 192.168.0.149 192.168.0.135	192.168.0.135 192.168.0.149

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

using namespace std;

int main() {
    int n;
    cin >> n;
    string* ips = new string[n];
    for (int i = 0; i < n; ++i) {
        cin >> ips[i];
    }
    for (int i = 0; i < n; ++i) {
        bool found = false;
        for (int j = 0; j < i && !found; ++j) {
            if (ips[j] == ips[i]) found = true;
        }
        if (!found) {
            for (int j = i + 1; j < n && !found; ++j) {
                if (ips[j] == ips[i]) {
                    found = true;
                    cout << ips[i] << endl;
                }
            }
        }
    }
    delete[] ips;
    return 0;
}
```

Egy ***Lakas*** struktúra az alábbi módon van definiálva:

```
struct Lakas {
  int id; // Hirdetes azonositoja
  unsigned int terulet; // Alapterulet m^2-ben
  unsigned int ar; // Meghirdetett eladasi ar Ft-ban
};
```

Készítsen egy ***rendez*** függvényt, ami paraméterként vár egy ***Lakas*** tömböt, és a tömb elemszámát, majd rendezi a tömb elemeit négyzetméterenkénti ár szerint növekvő sorrendbe.

A tesztkódokban található kiir függvényt nem kell elkészíteni.

For example:

Test	Result

`Lakas lakasok[] = {`
`  {1, 76, 43500000},`
`  {2, 100, 38800000},`
`  {3, 62, 29900000},`
`  {4, 42, 22800000},`
`  {5, 125, 48500085}`
`};`
`rendez(lakasok, sizeof(lakasok)/sizeof(lakasok[0]));`
`kiir(lakasok, sizeof(lakasok)/sizeof(lakasok[0]));`
	`ID Terulet  Ar`
	`2 100 m^2  38800000 F`
	`5 125 m^2  48500085 F`
	`3  62 m^2  29900000 F`
	`4  42 m^2  22800000 F`
	`1  76 m^2  43500000 F`

```
void rendez(Lakas tomb[], int db) {
    for (int i = 0; i < db - 1; i++) {
        int mini = i;
        double minval = double(tomb[i].ar) / tomb[i].terulet;
        for (int j = i + 1; j < db; ++j) {
            if (double(tomb[j].ar) / tomb[j].terulet < minval) {
                mini = j;
                minval = tomb[j].ar / tomb[j].terulet;
            }
        }
        if (mini != i) {
            Lakas csere = tomb[i];
            tomb[i] = tomb[mini];
            tomb[mini] = csere;
        }
    }
}
```

Egy autóverseny versenyzőinek aktuális sorrendjét egy láncolt lista tárolja, melynek elemei a következő struktúrával vannak definiálva:

```
struct Versenyzo {
  string nev;
  Versenyzo* kov;
};
```

Készítsen egy ***eloz*** nevű függvényt, ami megkeres egy adott nevű versenyzőt, és felcseréli a listában előtte lévő elemmel (ha van ilyen).

A függvény 1. paramétere a lista első elemének címe (a horgony), 2. paramétere a keresendő név. A függvény térjen vissza a művelet utáni lista első elemének címével.

Ha a keresett nevű versenyző az 1. helyen van, vagy nem található, a függvény ne változtasson a listán! Feltételezheti, hogy a versenyzők neveiben nincs egyezés.

Tipp: *Nem kell a lista elemeit ténylegesen felcserélni, elég a versenyzők neveit kicserélni.*

For example:

Test	Result
`Versenyzo* horgony = eloz(NULL, "Magnussen");` `kiir(horgony);`	

`Versenyzo* horgony = NULL;`
`beszur(horgony, "Schumacher");`
`kiir(horgony);`
`horgony = eloz(horgony, "Schumacher");`
`kiir(horgony);`
`horgony = eloz(horgony, "Hamilton");`
	`Schumacher`
	
	`Schumacher`
	
	`Schumacher`

```
Versenyzo* eloz(Versenyzo* horgony, string nev) {
    if (!horgony || !horgony->kov) return horgony;
    Versenyzo* akt = horgony;
    while (akt->kov && akt->kov->nev != nev) akt = akt->kov;
    if (akt->kov) {
        string tmp = akt->nev;
        akt->nev = akt->kov->nev;
        akt->kov->nev = tmp;
    }
    return horgony;
}
```

Egy ***matrix*** struktúra az alábbi módon van definiálva:

```
struct matrix {
  int sorok; // A matrix sorainak szama
  int oszlopok; // A matrix oszlopainak szama
  double** m; // Matrix mutatotombos alakban megadva; 'sorok' darab 'double*'
};
```

Készítsen egy ***jobbraForgat*** függvényt, ami paraméterben kap egy ilyen mátrix struktúrát, és visszaadja annak 90°-kal jobbra forgatott másolatát, azaz az 1. sorból lesz az utolsó oszlop. Az új mátrixnak dinamikusan foglaljon megfelelő méretű memóriaterületet!

A tesztkódokban található beolvasás és felszabadítás függvényeket nem kell elkészíteni. A mátrix dimenziói mindig nemnegatív számok, és a mátrix méretei ennek megfelelőek, ezek ellenőrzésével nem kell foglalkozni.

```
matrix jobbraForgat(const matrix& mtx) {
    matrix uj = {mtx.oszlopok, mtx.sorok, nullptr};
    uj.m = new double*[uj.sorok];
    for (int i = 0; i < uj.sorok; ++i) {
        uj.m[i] = new double[uj.oszlopok];
        for (int j = 0; j < uj.oszlopok; ++j) {
            uj.m[i][j] = mtx.m[mtx.sorok - j - 1][i];
        }
    }
    return uj;
}
```