Metronome

```cpp
#include <iostream>
using namespace std;

// time_1 and time_2 are two consecutive values in the input chanel
// returns the elapsed time between time_1 and time_2
int time_lapse(int time_1, int time_2) {
    if (time_2 > time_1) return time_2 - time_1;
    else return 60 + time_2 - time_1;
}

int main() {
    int pre_time;
    cin >> pre_time ;
    int time;
    cin >> time;
    int lapse = time_lapse(pre_time, time);
    while (time != -1 and time_lapse(pre_time, time) == lapse) {
        pre_time = time;
        cin >> time;
    }
    if (time == -1) cout << lapse;
    else cout << 0;
    cout << endl;
}
```

Zero count

```
#include <iostream>
#include <vector>
using namespace std;

//pre: ---
//post: u and v have the same size. For each position j value v[j]
//      is the number of zeros in u[j...last_position]
void zeros_counter(const vector<int>& u, vector<int>& v) {
    int n = u.size();
    v = vector<int>(n);
    int counter = 0;
    for (int j = n - 1; j >= 0; --j) {
        if (u[j] == 0) ++counter;
        v[j] = counter;
    }
}

int main() {
    int n;
    cin >> n;
    vector<int> u(n);
    for (int i = 0; i < n; ++i) cin >> u[i];
    vector<int> v;
    zeros_counter(u, v);
    for (int i = 0; i < n; ++ i) cout << v[i] << endl;
}
```

IMPORTANTE: Las soluciones que no implementen el procedimiento se consideran INVÁLIDAS.

Room with objects

```
#include <iostream>
#include <vector>
using namespace std;

struct Item {
    string name;
    int quantity;
};

typedef vector<vector<Item> > Room;

// Pre: n, m integers greater than 0
// Post: it returns a valid n*m Room
Room read_room(int n, int m) {
    Room r(n, vector<Item>(m));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) cin >> r[i][j].name >> r[i][j].quantity;
    }
    return r;
}

// Pre: room has at least one object f, c is a valid position of room.
// Post: it returns the total amount of objects in room having name s
//       and included in the submatrix having its top left corner at
//       position f, c, and its bottom right corner as in room
int how_many_objects(const Room& room, int f, int c, string s) {
    int q = 0;
    for (int i = f; i < room.size(); ++i) {
        for (int j = c; j < room[0].size(); ++j) {
            if (room[i][j].name == s) q = q + room[i][j].quantity;
        }
    }
    return q;
}

int main() {
    int n, m;
    cin >> n >> m;
    Room room = read_room(n, m);
    int f, c;
    string s;
    while (cin >> f >> c >> s) {
        cout << s << ": " << how_many_objects(room, f, c, s) << endl;
```

```
    }
}
```

Efficient search

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

//pre: ----
//post: returns true when a is shorter than b or when they have
//      the same length but a is less than b in the usual
//      string order. Returns false otherwise.
bool is_lower(string a, string b) {
    if (a.length() == b.length()) return a < b;
    return a.length() < b.length();
}

//pre:  v is ordered according to string length first and then
//      by usual string order. All strings are different.
//post: returns the position of s in v when s is in v.
//      If s  is not in v, returns -1
int effi_search(const vector<string>& v, string  s) {
    int iz = 0;
    int de = v.size() - 1;
    while (iz <= de) {
        int mid = (iz + de)/2;
        if (v[mid] == s) return mid;
        else if (is_lower(v[mid], s)) iz = mid + 1;
        else de = mid - 1;
    }
    return -1;
}

int main() {
    int n;
    cin >> n;
    vector<string> v(n);
    for (int i = 0; i < n; ++i) cin >> v[i];
    string s;
    while (cin >> s)
        cout << effi_search(v, s) << endl;
}
```