Dynamic Array

```
#include <bits/stdc++.h>

using namespace std;

string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);

/*
 * Complete the 'dynamicArray' function below.
 *
 * The function is expected to return an INTEGER_ARRAY.
 * The function accepts following parameters:
 *  1. INTEGER n
 *  2. 2D_INTEGER_ARRAY queries
 */

vector<int> dynamicArray(int n, vector<vector<int>> queries) {
    vector<vector<int>> arr(n);
    int lastAnswer = 0;
    vector<int> answers;

    for (int i = 0; i < queries.size(); i++) {
        int type = queries[i][0];
        int x = queries[i][1];
        int y = queries[i][2];

        int idx = (x ^ lastAnswer) % n;

        if (type == 1) {
            arr[idx].push_back(y);
        }
        else if (type == 2) {
            int pos = y % arr[idx].size();
            lastAnswer = arr[idx][pos];
            answers.push_back(lastAnswer);
        }
    }

    return answers;
}

int main()
```

```
{
    ofstream fout(getenv("OUTPUT_PATH"));

    string first_multiple_input_temp;
    getline(cin, first_multiple_input_temp);

    vector<string> first_multiple_input =
split(rtrim(first_multiple_input_temp));

    int n = stoi(first_multiple_input[0]);

    int q = stoi(first_multiple_input[1]);

    vector<vector<int>> queries(q);

    for (int i = 0; i < q; i++) {
        queries[i].resize(3);

        string queries_row_temp_temp;
        getline(cin, queries_row_temp_temp);

        vector<string> queries_row_temp =
split(rtrim(queries_row_temp_temp));

        for (int j = 0; j < 3; j++) {
            int queries_row_item = stoi(queries_row_temp[j]);

            queries[i][j] = queries_row_item;
        }
    }

    vector<int> result = dynamicArray(n, queries);

    for (size_t i = 0; i < result.size(); i++) {
        fout << result[i];

        if (i != result.size() - 1) {
            fout << "\n";
        }
    }

    fout << "\n";

    fout.close();

    return 0;
}

string ltrim(const string &str) {
    string s(str);

    s.erase(
        s.begin(),
        find_if(s.begin(), s.end(), not1(ptr_fun<int,
int>(isspace)))
    );

    return s;
}

string rtrim(const string &str) {
    string s(str);

    s.erase(
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int,
int>(isspace))).base(),
        s.end()
    );

    return s;
}

vector<string> split(const string &str) {
    vector<string> tokens;

    string::size_type start = 0;
    string::size_type end = 0;

    while ((end = str.find(" ", start)) != string::npos) {
        tokens.push_back(str.substr(start, end - start));

        start = end + 1;
    }

    tokens.push_back(str.substr(start));

    return tokens;
}
```