#include <assert.h>

#include <ctype.h>

#include <limits.h>

#include <math.h>

#include <stdbool.h>

#include <stddef.h>

#include <stdint.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

char\* readline();

char\* ltrim(char\*);

char\* rtrim(char\*);

char\*\* split\_string(char\*);

int parse\_int(char\*);

/\*

 \* 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

 \*/

/\*

 \* To return the integer array from the function, you should:

 \*     - Store the size of the array to be returned in the result\_count variable

 \*     - Allocate the array statically or dynamically

 \*

 \* For example,

 \* int\* return\_integer\_array\_using\_static\_allocation(int\* result\_count) {

 \*     \*result\_count = 5;

 \*

 \*     static int a[5] = {1, 2, 3, 4, 5};

 \*

 \*     return a;

 \* }

 \*

 \* int\* return\_integer\_array\_using\_dynamic\_allocation(int\* result\_count) {

 \*     \*result\_count = 5;

 \*

 \*     int \*a = malloc(5 \* sizeof(int));

 \*

 \*     for (int i = 0; i < 5; i++) {

 \*         \*(a + i) = i + 1;

 \*     }

 \*

 \*     return a;

 \* }

 \*

 \*/

int\* dynamicArray(int n, int queries\_rows, int queries\_columns, int\*\* queries, int\* result\_count) {

    int\*\* arr=(int\*\*)malloc(n\*sizeof(int\*));

    int\* sizes=(int\*)calloc(n,sizeof(int));

    int\* capacities=(int\*)calloc(n,sizeof(int));

    for(int i=0;i<n;i++){

        arr[i]=NULL;

        sizes[i]=0;

        capacities[i]=0;

    }

    int lastAnswer=0;

    int\* results=(int\*)malloc(queries\_rows\*sizeof(int));

    int res\_idx=0;

    for(int i=0;i<queries\_rows;i++){

        int type=queries[i][0];

        int x=queries[i][1];

        int y=queries[i][2];

        int idx=(x^lastAnswer)%n;

        if(type==1){

            if(sizes[idx]==capacities[idx]){

                capacities[idx]=capacities[idx]==0?2:capacities[idx]\* 2;

                arr[idx]=(int\*)realloc(arr[idx],capacities[idx] \* sizeof(int));

            }

            arr[idx][sizes[idx]++]=y;

        }

        else if(type==2){

            lastAnswer=arr[idx][y%sizes[idx]];

            results[res\_idx++]=lastAnswer;

        }

    }

    \*result\_count=res\_idx;

    return results;

}

int main()

{

    FILE\* fptr = fopen(getenv("OUTPUT\_PATH"), "w");

    char\*\* first\_multiple\_input = split\_string(rtrim(readline()));

    int n = parse\_int(\*(first\_multiple\_input + 0));

    int q = parse\_int(\*(first\_multiple\_input + 1));

    int\*\* queries = malloc(q \* sizeof(int\*));

    for (int i = 0; i < q; i++) {

        \*(queries + i) = malloc(3 \* (sizeof(int)));

        char\*\* queries\_item\_temp = split\_string(rtrim(readline()));

        for (int j = 0; j < 3; j++) {

            int queries\_item = parse\_int(\*(queries\_item\_temp + j));

            \*(\*(queries + i) + j) = queries\_item;

        }

    }

    int result\_count;

    int\* result = dynamicArray(n, q, 3, queries, &result\_count);

    for (int i = 0; i < result\_count; i++) {

        fprintf(fptr, "%d", \*(result + i));

        if (i != result\_count - 1) {

            fprintf(fptr, "\n");

        }

    }

    fprintf(fptr, "\n");

    fclose(fptr);

    return 0;

}

char\* readline() {

    size\_t alloc\_length = 1024;

    size\_t data\_length = 0;

    char\* data = malloc(alloc\_length);

    while (true) {

        char\* cursor = data + data\_length;

        char\* line = fgets(cursor, alloc\_length - data\_length, stdin);

        if (!line) {

            break;

        }

        data\_length += strlen(cursor);

        if (data\_length < alloc\_length - 1 || data[data\_length - 1] == '\n') {

            break;

        }

        alloc\_length <<= 1;

        data = realloc(data, alloc\_length);

        if (!data) {

            data = '\0';

            break;

        }

    }

    if (data[data\_length - 1] == '\n') {

        data[data\_length - 1] = '\0';

        data = realloc(data, data\_length);

        if (!data) {

            data = '\0';

        }

    } else {

        data = realloc(data, data\_length + 1);

        if (!data) {

            data = '\0';

        } else {

            data[data\_length] = '\0';

        }

    }

    return data;

}

char\* ltrim(char\* str) {

    if (!str) {

        return '\0';

    }

    if (!\*str) {

        return str;

    }

    while (\*str != '\0' && isspace(\*str)) {

        str++;

    }

    return str;

}

char\* rtrim(char\* str) {

    if (!str) {

        return '\0';

    }

    if (!\*str) {

        return str;

    }

    char\* end = str + strlen(str) - 1;

    while (end >= str && isspace(\*end)) {

        end--;

    }

    \*(end + 1) = '\0';

    return str;

}

char\*\* split\_string(char\* str) {

    char\*\* splits = NULL;

    char\* token = strtok(str, " ");

    int spaces = 0;

    while (token) {

        splits = realloc(splits, sizeof(char\*) \* ++spaces);

        if (!splits) {

            return splits;

        }

        splits[spaces - 1] = token;

        token = strtok(NULL, " ");

    }

    return splits;

}

int parse\_int(char\* str) {

    char\* endptr;

    int value = strtol(str, &endptr, 10);

    if (endptr == str || \*endptr != '\0') {

        exit(EXIT\_FAILURE);

    }

    return value;

}