#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\*);

void sieve\_primes(int primes[],int q){

    int limit=100000;

    bool \*is\_prime=(bool\*)malloc((limit+1)\* sizeof(bool));

    for(int i=0;i<=limit;i++)

    is\_prime[i]=true;

    is\_prime[0]=is\_prime[1]=false;

    int count=0;

    for(int p=2;p<=limit && count<q;p++){

        if(is\_prime[p]){

            primes[count++]=p;

            for(int j=p\*2;j<=limit;j+=p)

            is\_prime[j]=false;

        }

    }

    free(is\_prime);

}

/\*

 \* Complete the 'waiter' function below.

 \*

 \* The function is expected to return an INTEGER\_ARRAY.

 \* The function accepts following parameters:

 \*  1. INTEGER\_ARRAY number

 \*  2. INTEGER q

 \*/

/\*

 \* 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\* waiter(int\* numbers, int n, int q, int\* result\_count) {

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

sieve\_primes(primes,q);

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

int ans\_size=0;

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

int a\_size=n;

for(int i=0;i<n;i++) A[i]=numbers[i];

for(int i=0;i<q;i++){

    int p=primes[i];

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

    int nextA\_size=0;

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

    int b\_size=0;

    for(int j=a\_size-1;j>=0;j--){

        if(A[j]%p==0){

            B[b\_size++]=A[j];

        }else{

            nextA[nextA\_size++]=A[j];

        }

    }

    for(int j=b\_size-1;j>=0;j--){

        ans[ans\_size++]=B[j];

    }

    free(A);

    free(B);

    A=nextA;

    a\_size=nextA\_size;

}

for(int j=a\_size-1;j>=0;j--){

    ans[ans\_size++]=A[j];

}

\*result\_count=ans\_size;

free(A);

free(primes);

return ans;

}

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));

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

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

    for (int i = 0; i < n; i++) {

        int number\_item = parse\_int(\*(number\_temp + i));

        \*(number + i) = number\_item;

    }

    int result\_count;

    int\* result = waiter(number, n, q, &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;

}