#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\*);

int parse\_int(char\*);

/\*

 \* Complete the 'caesarCipher' function below.

 \*

 \* The function is expected to return a STRING.

 \* The function accepts following parameters:

 \*  1. STRING s

 \*  2. INTEGER k

 \*/

/\*

 \* To return the string from the function, you should either do static allocation or dynamic allocation

 \*

 \* For example,

 \* char\* return\_string\_using\_static\_allocation() {

 \*     static char s[] = "static allocation of string";

 \*

 \*     return s;

 \* }

 \*

 \* char\* return\_string\_using\_dynamic\_allocation() {

 \*     char\* s = malloc(100 \* sizeof(char));

 \*

 \*     s = "dynamic allocation of string";

 \*

 \*     return s;

 \* }

 \*

 \*/

char\* caesarCipher(char\* s, int k) {

int n=strlen(s);

k=k%26;

char\* result=(char\*)malloc((n+1)\*sizeof(char));

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

    char c=s[i];

    if(isalpha(c)){

        if(isupper(c)){

            result[i]='A'+(c-'A'+k)%26;

        }else{

            result[i]='a'+(c-'a'+k)%26;

        }

    }else{

        result[i]=c;

    }

}

result[n]='\0';

return result;

}

int main()

{

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

    int n = parse\_int(ltrim(rtrim(readline())));

    char\* s = readline();

    int k = parse\_int(ltrim(rtrim(readline())));

    char\* result = caesarCipher(s, k);

    fprintf(fptr, "%s\n", result);

    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;

}

int parse\_int(char\* str) {

    char\* endptr;

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

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

        exit(EXIT\_FAILURE);

    }

    return value;

}