org 100h

inputString va fi parcurs cu pointerul SI, iar sirul OutputString va fi parcurs cu DI

init: lea SI, inputString // SI este initializat cu adresa primului element din inputString

lea DI, outputString //DI este initializat cu adresa primei locatii de memorie din outputString

mov CX, charX-inputString //CX este initializat cu numarul de elemente din inputString

cld // steagul DF(direction flag) este initializat cu – inseamna ca sirul va fi parcus de la stanga la dreapta

//Blocul compare parcurge sirul inputString si compara elementul curent cu charX. Daca elementul curent este charX, programul copiaza charY in outputString, altfel el copiaza elementul curent din sirul inputString in outputString.

compare: lodsb // Instructiunea loadsb copiaza elementul curent din sirul inputString in acumulator, in AL, si incremeneaza SI

cmp AL,[charX] // compara acumulatorul AL, cu valoarea din charX

jne copyCurrent // Daca cele doua nu sunt egale, copiaza elementul curent din AL

je copyCharY //Daca AL, pozitia curenta este egala cu valoarea lui charX, copiem charY

nextElem:loop compare // decrementeaza CX si daca CX!=0, sare la compare. Daca CX =0, sirul s-a terminat

copyBack: dec SI //

dec DI

xchg SI, DI

mov CX, charX-inputString

std

repnz movsb

end: int 20h

copyCurrent: stosb // incrementeaza DI , stocheaza valoarea din acumulator in outputString

jmp nextElem //sare inapoi la bucla compare

copyCharY: mov AL,[charY]

stosb

jmp nextElem

inputString db 'o iubesc pe mami' // declararea sirului inputString, pe 8 biti. Db=directiva de asamblare define byte

charX db 'i' // declararea caracterului pe care il cautam in sir pentru a il inlocui

charY db 'n' //declararea caracterului cu care il vom inlocui pe charX

outputString db 20 dup('X') // aloca un sir de 20 de locatii de memorie si initializeaza toate valorile din sir cu X mare. Acesta va fi sirul de iesire.