**Laborator 9 - Suport teoretic**

**Programare multi-modul (asm+asm)**

**Programare multi-modul (asm+asm)**

Programele non-triviale (de amploare) prezintă problematici specifice complexității crescute a codului, necesitând ca atare și instrumente care adresează aceste aspecte.  Natural apar următoarele întrebări:

* cum este posibil a fi împărțită (descompusă) problema atacată în sub-probleme de dificultate redusă?
* care dintre sub-problemele identificate după descompunere sunt deja cunoscute și au rezolvări consacrate, bine cunoscute și care pot fi refolosite?

## Subprograme în limbajul de asamblare x86

* O variantă pentru împărțirea codului în sub-probleme o reprezintă modularizarea codului. Limbajul de asamblare nu recunoaște noțiunea de subprogram. Putem însă crea o secvență de instrucţiuni care să poată fi apelată din alte zone ale programului și după terminarea ei să returneze controlul programului apelant.
* O astfel de secvență se numește procedură. Apelul unei proceduri se poate face cu o instrucţiune jmp. Problema care apare la un astfel de salt este că procesorul nu ține minte de unde a fost trimis la "procedură" şi prin urmare nu știe unde sa revină cu execuţia după terminarea procedurii. Este necesar deci ca la apelul unei proceduri să salvăm undeva adresa de revenire, iar revenirea din procedură este de fapt o instrucţiune de salt la acea adresă.
* Locul unde se salvează adresa de revenire este stivă de execuţie. Este nevoie de stivă deoarece o procedură poate apela o altă procedură, acea procedură poate apela alta, s.a.m.d.
* Există două instrucţiuni ce permit apelul și revenirea din proceduri: call şi ret.

### Sintaxa:

call eticheta

* Instrucţiunea call este de fapt o instrucţiune jmp care în plus introduce în vârful stivei adresa instrucţiunii care urmează – valoarea din EIP (instrucţiunea care vine imediat după call și nu destinația saltului produs de instrucţiunea call).
* Instrucţiunea ret extrage din vârful stivei o adresă şi execută un salt la acea adresă (practic se modifică EIP, valoarea extrasă de pe stivă este stocată în registrul EIP). Instrucţiunea nu are argumente deoarece adresa la care sare programul este extrasă din vârful stivei (nu este dată explicit).

### Observație:

* Pentru a evita posibilele ambiguități ce pot să apară atunci când numele unei etichete definită în procedură (etichetă locală) este identic cu numele altei etichete definite în programul "principal", numele etichetelor "locale" trebuie să înceapă cu caracterul "."

### Exemplu

.eticheta:; etichetă utilizată în procedură

eticheta:; etichetă globală

* Asamblorul  NASM oferă prin intermediul preprocesorului un mecanism simplu prin care un program de multe linii poate fi împărțit în mai multe fişiere. În general, datorită similarității cu directiva #include a limbajului C, rolul de utilizare în practică al acesteia este același: permite separarea declarațiilor programului în unul sau mai multe fişiere ce vor fi incluse acolo unde respectivele declarații sunt necesare, întocmai cum și în C se obișnuiește separarea/gruparea declarațiilor în fișiere header (fişiere cu extensia .h), fişiere ce sunt ulterior incluse de către codul scris în fișiere C care necesită (referă) declarațiile în cauză.
* La nivel de preprocesor, directiva **%include** îi permite unei componente de program să fie construită din mai multe fișiere ce vor fi asamblate împreună.
* Ca urmare o procedura poate fi definită în același fişier sursa (a se vedea lab11\_procedura.asm) sau într-un fişier separat (a se vedea lab11\_proc\_main.asm şi factorial.asm, ambele fişiere sunt în același director, asamblarea, editarea de legături şi execuţia se fac ca şi până acum). În aceste cazuri, în urma asamblării veți obține un singur fișier \*.obj!

### *lab11\_procedura.asm*

; programul calculează factorialul unui număr şi afişează în consolă rezultatul

; procedura factorial este definita in segmentul de cod şi primește pe stivă ca parametru un număr

bits 32

global start

extern printf, exit

import printf msvcrt.dll

import exit msvcrt.dll

segment data use32 class=data

format\_string db "factorial=%d", 10, 13, 0

segment code use32 class=code

; urmeaza definirea procedurii

factorial:

mov eax, 1

mov ecx, [esp + 4]

; mov ecx, [esp + 4] scoate de pe stivă parametrul procedurii

; ATENTIE!!! in capul stivei este adresa de retur

; parametrul procedurii este imediat după adresa de retur

; a se vedea desenul de mai jos

;

; stivă

;

;|-------------------|

;| adresa retur  |  <- esp

;|-------------------|

;|  00000006h  |  <- parametrul pasat procedurii, esp+4

;|-------------------|

; ....

.repet:

mul ecx

loop .repet ; atenție, cazul ecx = 0 nu e tratat!

ret

; programul "principal"

start:

push dword 6        ; se salvează pe stivă numărul (parametrul procedurii)

call factorial      ; apel procedura

; afişare rezultat

push eax

push format\_string

call [printf]

add esp, 4\*2

push 0

call [exit]

### *lab11\_proc\_main.asm - Diferența față de lab11\_procedura.asm este ca procedura factorial este definită în alt fişier (factorial.asm) fiind necesara includerea acestuia folosind directiva %include.*

; programul calculează factorialul unui număr şi afişează in consola rezultatul

; procedura factorial este definită în fişierul factorial.asm

bits 32

global start

import printf msvcrt.dll

import exit msvcrt.dll

extern printf, exit

; codul definit in factorial.asm va fi copiat aici

%include "factorial.asm"

segment data use32 class=data

format\_string db "factorial=%d", 10, 13, 0

segment code use32 class=code

start:

push dword 6

call factorial

push eax

push format\_string

call [printf]

add esp, 2\*4

push 0

call [exit]

### *factorial.asm*

%ifndef \_FACTORIAL\_ASM\_ ; continuam daca \_FACTORIAL\_ASM\_ este nedefinit

%define \_FACTORIAL\_ASM\_ ; şi ne asiguram ca devine definit

; astfel %include permite doar o singura includere

;definire procedura

factorial:  ; int \_stdcall factorial(int n)

mov eax, 1

mov ecx, [esp + 4]

; mov ecx, [esp + 4] scoate de pe stivă parametrul procedurii

; pentru explicații a se vedea lab11\_procedura.asm

repet:

mul ecx

loop repet ; atenție, cazul ecx = 0 nu e tratat!

ret 4

%endif

**Programe din mai multe module**

Un program scris în limbaj de asamblare poate fi împărțit în mai multe fişiere sursă, fiecare fiind asamblat separat în fişiere *.obj*. Pentru a scrie un program din mai multe fişiere sursă trebuie să respectăm următoarele:

* toate segmentele vor fi declarate cu modificatorul *public*, pentru că în programul final segmentul de cod este construit din concatenarea segmentelor de cod din fiecare modul; la fel şi segmentul de date.
* etichetele şi numele variabilelor dintr-un modul care trebuie "exportate" în alte module trebuie să facă obiectul unor declarații *global*
* etichetele şi variabilele care sunt declarate într-un modul şi sunt folosite în alt modul trebuie să fie "importate" prin directiva *extern*
* o variabila trebuie declarată în întregime într-un modul (nu poate fi jumătate într-un modul şi jumătate într-altul). De asemenea, trecerea execuţiei dintr-un modul în altul se poate face doar prin instrucţiuni de salt (*jmp*, *call* sau *ret*).
* punctul de intrare este prezent doar în modulul ce conține "programul principal"

Fiecare modul se va asambla separat, folosind comanda:

nasm -fobj nume\_modul.asm

apoi modulele se vor lega împreună cu comanda

alink modul\_1.obj modul\_2.obj ... modul\_n.obj -oPE -subsys console -entry start

## Etape asamblare / link-editare/ debugging / execuţie

### ASAMBLARE:

nasm -f obj modul.asm

* Opțiunea -f indică tipul de fişier care să fie generat, în cazul acesta fişier obj.

### LINK-EDITARE

alink modul1.obj ... moduln.obj -o PE -subsys console -entry start

* În folder-ul nasm din asm\_tools exista fişierul "ALINK.TXT" care descrie opțiunile pentru alink.
* Alink options:

-o xxx

xxx:

* + COM = output COM file
  + EXE = output EXE file
  + PE = output Win32 PE file (.EXE)

-subsys xxx:

Opţiunea specifică tipul de aplicație generată (default=windows)

* + windows, win or gui => windows subsystem
  + console, con or char => console subsystem
  + native => native subsystem
  + posix => POSIX subsystem

-entry name

Opțiunea specifică punctul de intrare în program (prima instrucţiune de executat)

### DEBUG

OLLYDBG.EXE modul.exe

### EXECUŢIE

modul.exe

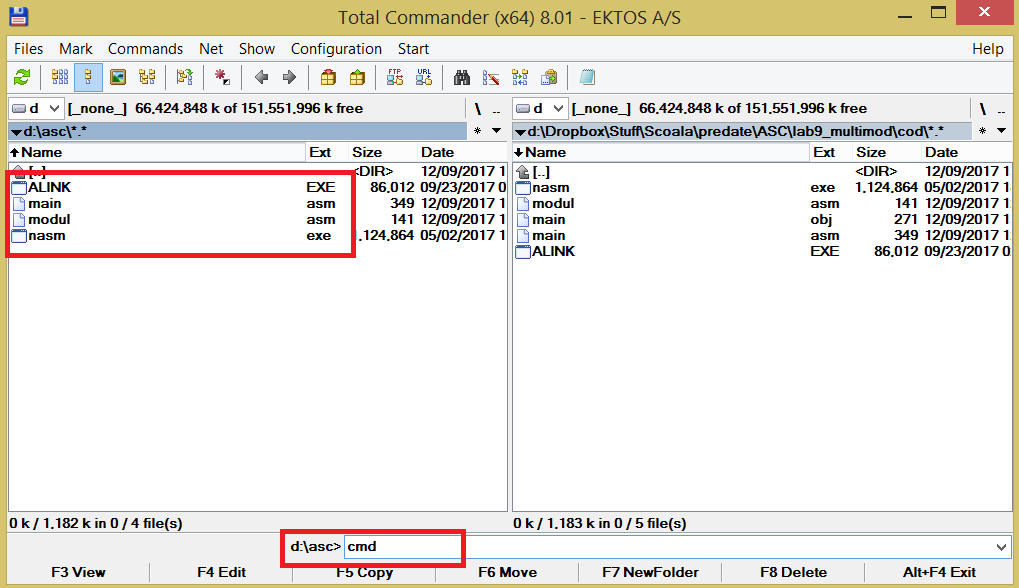
**Laborator 9 - Exemple**

**Programare multi-modul (asm+asm)**

**Exemple**

Se va scrie un program care calculează factorialul unui număr. "Programul principal" este *main.asm*, iar funcția factorial este definită în *modul.asm*. Pentru a putea asambla şi edita legături este nevoie de linia de comandă. Mai jos este prezentat procesul de asamblare, editare de legături ce duce la un program executabil:

* Fie directorul *asc* pe discul *D*, în acest director se găsesc sursele programului (*main.asm* şi *modul.asm*);
* Este necesar ca în directorul *asc* să se copieze și *nasm.exe* și *alink.exe* (programe disponibile în directorul *nasm* din setul de instrumente);
* Pentru a ușura navigarea în linia de comandă către locația pe disc unde se află sursele se poate folosi programul Total Commander, se navighează vizual până în directorul asc și se lansează linia de comandă (a se vedea figura de mai jos)

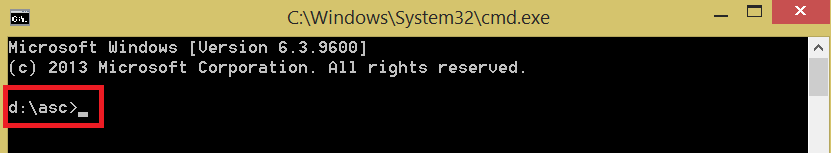


* dacă nu se vrea utilizarea programului Total Commander, se lansează cmd.exe și cu următoarea secvență de comenzi se poate naviga în directorul asc

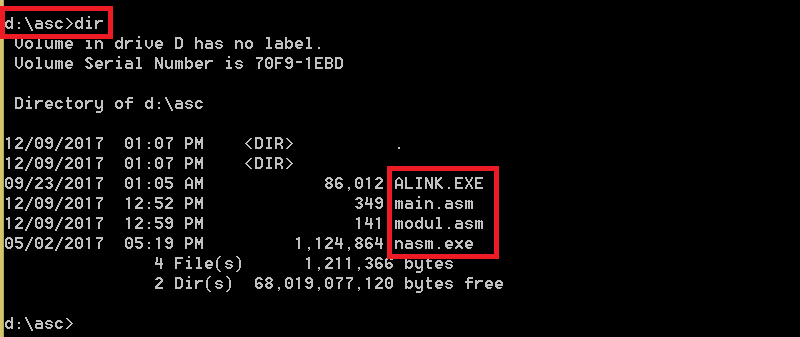
> d:

> cd asc

* indiferent de varianta aleasă rezultatul trebuie să fie cel din figura de mai jos



* comanda *dir* listează conținutul directorului curent, se verifică dacă directorul în care s-a navigat conține sursele *.asm* și programele *alink.exe* şi *nasm.exe* (a se vedea figura de mai jos)

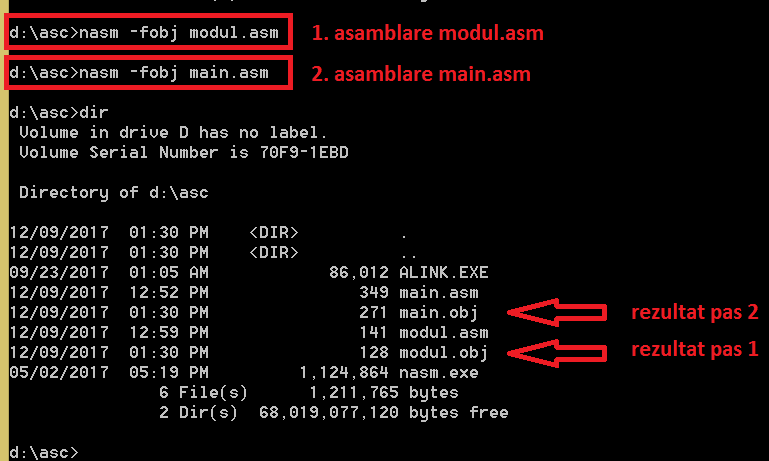


* se asamblează cele două surse folosind secvența de comenzi

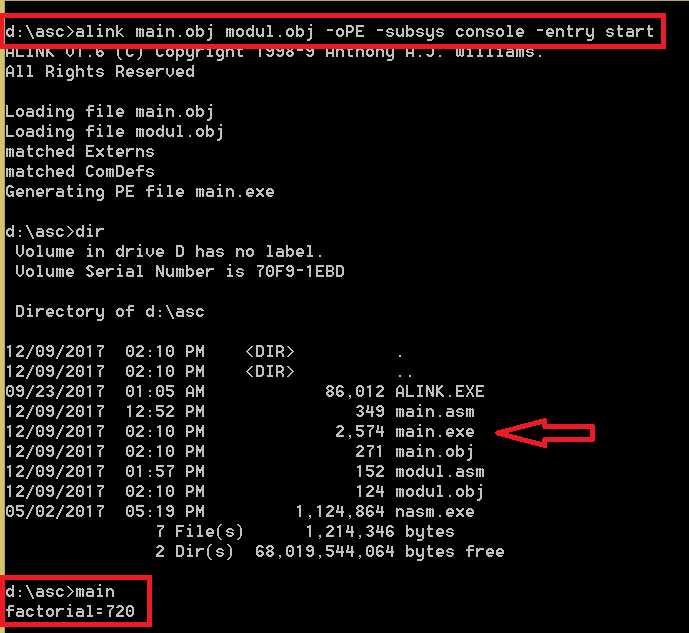
> nasm -fobj modul.asm

> nasm -fobj main.asm

* rezultatul este cel din figura de mai jos, două fişiere *.obj*



* folosind alink.exe (editor de legături), din cele două fișiere .obj se creează programul executabil, numele fişierului .exe rezultat este identic cu primul fişier .obj dat ca şi parametru editorului de legături, în acest caz *main*. Programul poate fi rulat, în acest caz *main.exe* afișează pe ecran 6! = 720 (a se vedea figura de mai jos)



* pentru a depana, codul din ollydbg, File->Open şi se deschide fişierul executabil, în acest caz main.exe

**Programe sursa**

Exemplul 1:

1. [lab11\_1.asm](http://www.cs.ubbcluj.ro/~vancea/asc/asc.2018-2019/files/lab11_1.asm)

Exemplul 2:

1. [lab11\_proc\_main.asm](http://www.cs.ubbcluj.ro/~vancea/asc/asc.2018-2019/files/lab11_proc_main.asm)
2. [factorial.asm](http://www.cs.ubbcluj.ro/~vancea/asc/asc.2018-2019/files/factorial.asm)

Exemplul 3:

1. [main.asm](http://www.cs.ubbcluj.ro/~vancea/asc/asc.2018-2019/files/main.asm)
2. [modul.asm](http://www.cs.ubbcluj.ro/~vancea/asc/asc.2018-2019/files/modul.asm)

## Factorial recursiv - Exemplu propus de studentul Molnar Radu, grupa 215, 2017-2018

bits 32 ; assembling for the 32 bits architecture

; declare the EntryPoint (a label defining the very first instruction of the program)

global start

; declare external functions needed by our program

extern exit,printf,scanf; adăugăm funcţiile externe de care avem nevoie

import exit msvcrt.dll

import printf msvcrt.dll

import scanf msvcrt.dll

; our data is declared here (the variables needed by our program)

segment data use32 class=data

; ...

text db "introduceți un n=",0

final db "n!=%d",0

format db "%d",0

a resd 1 ; variabila a va conține numărul n citit de la tastatură

; our code starts here

segment code use32 class=code

factor:

;pentru a implementa problema recursiv trebuie să o despărțim în cazuri

;la calcularea factorialului recursiv exista doua cazuri:

;n!=n\*(n-1)! - iterația curenta

;0!=1 - condiția de oprire

;subprogramul se va reapela pana in momentul în care se ajunge la ecx=0 moment in care se atribuie eax = 1 şi ne întoarcem la pasul anterior

mov ecx, [esp+4] ;mutam in ecx numărul de pași pe care îi mai avem de făcut

jecxz sf ;daca ecx ajunge sa fie 0 sărim la eticheta sf pentru a putea începe să calculăm factorialul

;dacă am trecut de compararea cu 0 atunci ajungem la primul caz al recursivității

;formula lui fiind n!=n\*(n-1)!

dec ecx; decrementăm ecx pentru a putea sa reapelăm funcția pentru pasul următor

push ecx; depunem pe stivă valoarea n curenta de calcul a factorialului

call factor; apelam funcția cu parametrul curent valoarea de pe stivă

mul dword [esp+8];înmulțim cu valoarea corespunzătoare pasului actual

add esp,4; eliberam stivă de parametrul utilizat pentru a ajunge la adresa de revenire a pasului anterior

jmp return; salt la eticheta return pentru a putea invoca revenirea din subprogram

sf:

mov eax,1;cum recursivitatea noastră are doua cazuri am ajuns in cazul in care ecx e 0 şi returnam 1 – condiția de oprire

;0!=1

return:

ret ;ne întoarcem la pasul anterior sau in programul principal

start:

; ...

;tipărirea mesajului

push dword text

call [printf]

add esp,4

;citirea lui n de la tastatura

push dword a

push dword format

call [scanf]

add esp,4\*2

mov ecx,0

mov eax,0;pregătim regiştrii pentru apelare

push dword [a] ;salvam pe stivă numărul n

call factor ;apelam funcția

;afișarea rezultatului

push eax

push final

call [printf]

; exit(0)

push dword 0; push the parameter for exit onto the stack

call [exit]; call exit to terminate the program

# Probleme propuse

## Programare multi-modul (asm+asm)

Pentru următoarele probleme se cere cel puțin un subprogram implementat într-un modul separat.

1. Se da un număr *a* reprezentat pe 32 biți fără semn. Se cere să se afişeze reprezentarea în baza 16 a lui *a*, precum şi rezultatul permutărilor circulare ale cifrelor sale.
2. Se cere sa se citeasca de la tastatura un sir de numere, date in baza 10 ca numere cu semn (se citeste de la tastatura un sir de caractere si in memorie trebuie stocat un sir de numere).
3. Se dau doua şiruri conținând caractere. Sa se calculeze si sa se afişeze rezultatul concatenării tuturor caracterelor tip cifra zecimala din cel de-al doilea sir dupa cele din primul sir si invers, rezultatul concatenarii primului sir dupa al doilea.
4. Se da un sir de numere. Sa se afişeze valorile in baza 16 si in baza 2.
5. Se cere se se citeasca numerele a, b si c ; sa se calculeze si sa se afişeze a+b-c.
6. Se citesc trei şiruri de caractere. Sa se determine si sa se afişeze rezultatul concatenarii lor.
7. Se dau trei şiruri de caractere. Sa se afişeze cel mai lung prefix comun pentru fiecare din cele trei perechi de cate doua şiruri ce se pot forma.
8. Sa se afişeze, pentru fiecare număr de la 32 la 126, valoarea numărului (in baza 8) si caracterul cu acel cod ASCII.
9. Se cere sa se citeasca de la tastatura un sir de numere, date in baza 16 (se citeste de la tastatura un sir de caractere si in memorie trebuie stocat un sir de numere). Sa se afişeze valoarea zecimala a nr atat ca numere fara semn cat si ca numere cu semn.
10. Se citesc mai multe şiruri de caractere. Sa se determine daca primul apare ca subsecventa in fiecare din celelalte si sa se dea un mesaj corespunzator.
11. Se citeste de la tastatura un sir de mai multe numere in baza 2. Sa se afişeze aceste numere in baza 16.
12. Se dau doua şiruri de caractere de lungimi egale. Se cere sa se calculeze si sa se afişeze rezultatele intercalarii literelor, pentru cele doua intercalari posibile (literele din primul sir pe pozitii pare, respectiv literele din primul sir pe pozitii impare).
13. Se dau trei şiruri de caractere. Sa se afişeze cel mai lung sufix comun pentru fiecare din cele trei perechi de cate doua şiruri ce se pot forma.
14. Se citesc mai multe numere de la tastatura, in baza 2. Sa se afişeze aceste numere in baza 8.
15. Se citesc de la tastatura un număr natural n si n propozitii care contin cel putin n cuvinte (nu se fac validari).  
    Sa se afişeze sirul format prin concatenarea cuvintelor de pe pozitia i din propozitia i, i=1,n (separate prin spatiu).  
    Exemplu: Se da: n=5  
    Ana are mere si pere.  
    Pe birou se gaseste un cos cu fructe.  
    Cartea mea preferata se afla pe masa.  
    Afara a nins si este destul de frig.  
    Maine o sa merg la cumparaturi.  
    Se va afişa:  
    Ana birou preferata si la
16. Se dau un sir care contine n reprezentari binare pe 8 biti date ca sir de caractere.  
    Sa se obtina un sir care contine numerele corespunzatoare reprezentarilor binare.  
    Exemplu:  
    Se da: '10100111b', '01100011b', '110b', '101011b'  
    Se stocheaza: 10100111b, 01100011b, 110b, 101011b
17. Se citeste de la tastatura un sir de numere in baza 10 fara semn. Sa se determine valoarea maxima din sir si sa se afişeze in fisierul max.txt (fisierul va fi creat) valoarea maxima, in baza 16
18. Se citeste de la tastatura un sir de numere in baza 10 fara semn. Sa se determine valoarea minima din sir si sa se afişeze in fisierul min.txt (fisierul va fi creat) valoarea minima, in baza 16
19. Se citesc din fisierul numere.txt mai multe numere (pozitive si negative). Sa se creeze 2 şiruri rezultat N si P astfel: N - doar numere negative si P - doar numere pozitive. Afişati cele 2 şiruri rezultate pe ecran.
20. Se citeste din fisierul numere.txt un sir de numere. Sa se determine sirul destinatie D care contine numerele din sirul initial cu valorile dublate dar in ordine inversa din sirul initial. Sa se afişeze sirul obtinut pe ecran.  
    Ex: s: 12, 2, 4, 5, 0, 7 => 14, 0, 10, 8, 4, 24
21. Se citeste de la tastatura un sir de numere in baza 10. Sa se afişeze numerele prime.
22. Se da un număr in baza 2 reprezentat pe 32 de biti. Sa se afişeze reprezentarea in baza 16. (se foloseste conversia rapida)
23. Sa se citeasca un sir de numere intregi s1 (reprezentate pe dublucuvinte) in baza 10. Sa se determine si sa se afişeze sirul s2 compus din cifrele aflate pe poziţia sutelor în fiecare număr intreg din sirul s1.  
    Exemplu:  
    Sirul s1: 5892, 456, 33, 7, 245  
    Sirul s2: 8, 4, 0, 0, 2
24. Sa se citeasca un sir s1 (care contine doar litere mici). Folosind un alfabet (definit in segmentul de date), determinati si afişati sirul s2 obtinut prin substituirea fiecarei litere a sirului s1 cu litera corespunzatoare din alfabetul dat.  
    Exemplu:  
    Alfabetul: OPQRSTUVWXYZABCDEFGHIJKLMN  
    Sirul s1:  anaaremere  
    Sirul s2:  OBOOFSASFS
25. Se citeste de la tastatura un sir de numere in baza 10, cu semn. Sa se determine valoarea maxima din sir si sa se afişeze in fisierul max.txt (fisierul va fi creat) valoarea maxima, in baza 16.
26. Se citeste de la tastatura un sir de numere in baza 10, cu semn. Sa se determine valoarea minima din sir si sa se afişeze in fisierul min.txt (fisierul va fi creat) valoarea minima, in baza 16.
27. Se citesc din fisierul numere.txt mai multe numere (pare si impare). Sa se creeze 2 şiruri rezultat N si P astfel: N - doar numere impare si P - doar numere pare. Afişati cele 2 şiruri rezultate pe ecran.
28. Se citeste de la tastatura un sir de caractere (litere mici si litere mari, cifre, caractere speciale, etc). Sa se formeze un sir nou doar cu literele mici si un sir nou doar cu literele mari. Sa se afişeze cele 2 şiruri rezultate pe ecran.
29. Se citeste o propozitie de la tastatura. Sa se numere literele din fiecare cuvant si sa se afişeze aceste numere pe ecran.
30. Se citeste o propozitie de la tastatura. Sa se inverseze fiecare cuvant si sa se afişeze pe ecran.

### Observatii

* Se da inseamna ca acele date pot fi puse direct in segmentul de date; se citesc inseamna ca acele date trebuie citite de la tastatura.
* Daca nu este precizat altfel, numerele se considera reprezentate pe 32 biti fara semn, iar şirurile de caractere de pana la 100 de caractere (sirul propriu-zis).