SEMINARUL 6 - ŞIRURI DE CARACTERE

1. Scrieţi o funcţie care să înlocuiască toate apariţiile unui şir *s* într-un şir *w* cu un şir *t*. De exemplu, dacă *w* = "dereferentierea", *s* = "ere" şi *t* = "as", şirul *w* va fi transformat în *w* = "dasfasntiasa".

Rezolvare:
O idee de rezolvare este următoarea: dacă şirul *s* apare în şirul *w* la adresa *p*, atunci salvăm într-o variabilă *aux* subşirul care urmează după *s* în *w* (adică subşirul care începe la adresa `p+strlen(s)`), copiem la adresa *p* şirul *t*, "refacem" şirul *w* concatenând la el subşirul salvat în *aux*, după care reluăm căutarea lui *s* în *w* după ultima adresă la care l-am găsit, respectiv `p+strlen(t)`.

```
void inlocuire(char *w, char *s, char *t)
{
   char *p, *aux;

      aux = (char*)malloc(strlen(w) + 1);

   p = strstr(w, s);
      while(p != NULL)
      {
        strcpy(aux, p + strlen(s));
        strcpy(p, t);
        strcat(w, aux);
        p = strstr(p + strlen(t), s);
    }

    free(aux);
}
```

Dacă am înlocui instrucţiunea `p=strstr(p+strlen(t),s)` cu instrucţiunea `p=strstr(p,s)`, funcţia ar mai furniza rezultatul corect în orice caz?

2. Scrieţi o funcţie care să se verifice dacă două şiruri de caractere sunt anagrame sau nu. Două şiruri sunt anagrame dacă unul se poate obţine din celălalt printr-o permutare a caracterelor sale. De exemplu, şirurile `emerit` şi `treime` sunt anagrame, dar şirurile `emerit` şi `treimi` nu sunt!

Rezolvare:
Problema poate fi rezolvată în mai multe moduri, dar, indiferent de metoda aleasă, se va testa mai întâi faptul că ambele şiruri au aceeaşi lungime:

a) Se observă faptul că două şiruri sunt anagrame dacă sunt formate din aceleaşi caractere şi fiecare caracter apare cu aceiaşi frecvenţă în ambele şiruri. Astfel, se construiesc 2 vectori de frecvenţe, câte unul pentru fiecare şir, şi apoi se verifică dacă aceştia sunt egali. În funcţie de tipul caracterelor din cele două şiruri, cei 2 vectori pot avea lungimi de 128/256 de elemente (ASCII/ASCII extins) sau o altă valoare convenabilă (de exemplu, dacă ştim faptul că ambele

şiruri sunt formate doar din litere mici, cei 2 vectori de frecvenţe pot avea doar 26 de elemente).

```
int anagrame_1(char *s, char *t)
{
    unsigned int i;
    unsigned int fs[256] = {0}, ft[256] = {0};

    if(strlen(s) != strlen(t))
        return 0;

    for(i = 0; i < strlen(s); i++)
    {
        fs[s[i]]++;
        ft[t[i]]++;
    }

    return memcmp(fs, ft, 256*sizeof(unsigned int)) == 0;
}
```

O altă variantă, asemănătoare, constă în utilizarea unui singur vector de frecvenţe, iar elementele sale vor fi incrementate pentru caracterele din primul şir şi apoi decrementate pentru caracterele din cel de-al doilea şir. Cum vom stabili, în acest caz, dacă şirurile sunt anagrame sau nu?

b) Se ordonează lexicografic caracterele din fiecare şir şi apoi se verifică egalitatea celor două noi şiruri, folosind funcţia `strcmp`. Atenţie, folosind această metodă, cele două şiruri iniţiale vor fi modificate!

```
int cmpCaractere(const void *a, const void* b)
{
    return *(char *)a - *(char *)b;
}

int anagrame_2(char *s, char *t)
{
    if(strlen(s) != strlen(t))
        return 0;

    qsort(s, strlen(s), sizeof(char), cmpCaractere);
    qsort(t, strlen(t), sizeof(char), cmpCaractere);

    return strcmp(s, t) == 0;
}
```

c) Se caută, pe rând, fiecare caracter din primul şir în cel de-al doilea. În cazul în care caracterul nu este găsit înseamnă că şirurile nu sunt anagrame, altfel se şterge caracterul din cel de-al doilea şir (de ce?) şi se trece la următorul caracter din primul şir. Atenţie, folosind această metodă, cel de-al doilea şir va fi modificat!

```
int anagrame_3(char *s, char *t)
```

```
{
    int i;
    char *p;

    if(strlen(s) != strlen(t))
        return 0;

    for(i = 0; i < strlen(s); i++)
    {
        p = strchr(t, s[i]);
        if(p == NULL)
            return 0;
        strcpy(p, p+1);
    }

    return 1;
}
```

Pentru fiecare dintre metodele prezentate mai sus, analizaţi complexitatea şi spaţiul de memorie utilizat!

3. Scrieţi un program care construieşte pentru fiecare angajat dintr-o companie o adresă de email de forma *prenume.nume@domeniu* si generează o parolă formată astfel: primele 4 caractere sunt litere mici ale alfabetului limbii engleze, iar următoarele 4 caractere sunt cifre. Datele de intrare se citesc din fişierul text *angajati.txt*. Fişierul conţine pe prima linie un şir de caractere reprezentând domeniul, iar pe următoarele linii numele şi prenumele fiecărui angajat din companie, separate printr-un spaţiu. Se consideră faptul că fişierul de intrare conţine doar primul prenume al unui angajat şi nu există doi angajaţi având numele şi prenumele identice. Datele fiecărui angajat, respectiv numele, prenumele, adresa de email, precum şi parola vor fi scrise în fişierul *conturi.csv*.

Rezolvare:

O metodă de rezolvare a acestei probleme constă în parsarea fiecărei linii din fişier în scopul de a extrage pentru fiecare angajat datele corespunzătoare, respectiv numele şi prenumele. Mai întâi, pentru fiecare angajat, se construieşte câte un şir de caractere `cont` de forma `prenume.nume@domeniu`. Deoarece fiecare linie din fişier conţine mai întâi numele şi apoi prenumele unui angajat, după fiecare apel al funcţiei `strtok`, vom extrage cele două date folosind variabilele de tip şir de caractere `nume`, respectiv `prenume`. Ulterior, pentru fiecare angajat în parte, se construieşte un şir `parola` care va memora parola unui angajat, formată din litere şi cifre generate aleator.

Pentru a genera un număr natural aleator se poate folosi funcţia `int rand()` din biblioteca `stdlib.h`, care furnizează o valoare cuprinsă între `0` şi `RAND_MAX`. Constanta `RAND_MAX` are o valoare implicită care poate să varieze de la un compilator la altul, dar se garantează faptul că este cel puţin 32767. Funcţia `rand()` este în sine o implementare a unui algoritm de generare a unui şir de numere pseudoaleatoare plecând de la o valoarea iniţială numită `seed`. Un generator de numere pseudoaleatoare are un comportament determinist, respectiv va genera acelaşi şir de numere dacă se utilizează aceeaşi valoare iniţială. În consecinţă, pentru a genera un şir de numere pseudoaleatore se stabileşte mai întâi valoarea `seed`, apelând funcţia `void srand`

`(unsigned int seed)` din biblioteca `stdlib.h`. Dacă funcţia `srand` nu este apelată înaintea funcţiei `rand`, atunci implicit valoarea iniţială a generatorului va avea valoarea 1. Deoarece parolele tuturor angajaţilor nu vor fi create neapărat printr-o singură rulare a programului, vom utiliza câte o valoare iniţială diferită pentru fiecare set de date de intrare, asigurându-ne astfel de caracterul aleator al tuturor parolelor. O soluţie în acest sens este utilizarea funcţiei `time(NULL)` care returnează numărul de secunde calculat începând de la data 1 ianuarie 1970, ora 00:00:00, până în momentul executării programului.

Primele patru caractere ale şirului parola, care sunt litere mici ale alfabetului englez, se obţin generând un număr aleator cuprins între 0 şi 26, la care se adună codul ASCII al caracterului `'a'`. În mod similar, ultimele 4 caractere ale şirului `parola`, care sunt cifre, se obţin generând un număr aleator cuprins între 0 şi 9 la care se adună codul ASCII al caracterului `'0'`.

Datele fiecărui angajat, respectiv numele, prenumele, adresa de email şi parola, sunt scrise în fişierul *conturi.csv*. Formtul de fişier text `CSV`(Comma-Separated Values) specifică faptul că informaţiile sunt stocate în formă tabelară. Acest tip de fişier poate fi deschis cu orice editor de text, dar pentru o vizualizare a datelor sub formă tabelară se recomandă utilizarea unei aplicaţii dedicate, cum ar fi Microsoft Excel. Implicit, separatorul de coloană este virgula.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

int main()
{
    char cont[100]="",dom[10],nume[50],prenume[50],linie[200], parola[9];
    char *p;
    FILE *f, *g;
    int x, i;

    f = fopen("angajati.txt", "r");
    g = fopen("conturi.csv", "w");

    if(f == NULL)
    {
        printf("Fisier inexistent!");
        return -1;
    }

    fgets(dom, 10, f);
    dom[strlen(dom)-1]='\0';

    srand(time(NULL));

    fprintf(g,"%s,%s,%s,%s\n","NUME","PRENUME","CONT EMAIL","PAROLA");

    while(fgets(linie, 500, f) != NULL)
    {
        p = strtok(linie," \n");
        strcpy(nume, p);
```

```
        nume[0] = nume[0] + ('a'-'A');

        p=strtok(NULL, "\n");
        strcpy(prenume, p);
        prenume[0] = prenume[0] + ('a' - 'A');

        strcpy(cont, prenume);
        strcat(cont, ".");
        strcat(cont, nume);
        strcat(cont,"@");
        strcat(cont, dom);

        for(i=0; i<4; i++)
        {
            x = rand()%26;
            parola[i] = 'a' + x;
        }

        for(i=4; i<8; i++)
        {
            x = rand()%10;
            parola[i] = '0' + x;
        }
        parola[8] = '\0';

        nume[0] = nume[0] - ('a'-'A');
        prenume[0] = prenume[0]- ('a'-'A');
        fprintf(g, "%s,%s,%s,%s\n", nume, prenume, cont, parola);
    }

    fclose(f);
    fclose(g);

    return 0;
}
```

4. Ana îşi notează în fişierul text *cumparaturi.txt* detalii despre cumpărăturile pe care le-a efectuat. Pentru fiecare produs cumpărat ea îşi notează cantitatea şi preţul unitar. Scrieţi un program care afişează pe ecran totalul cheltuielilor Anei.

Rezolvare:

Fişierul conţine pentru fiecare produs cumpărat mai întâi cantitatea, apoi preţul unitar. Astfel, o metodă de rezolvare a problemei presupune extragerea valorilor reale din fişier în variabila de tip real `x`, precum şi o contorizarea a numărului lor prin variabila de tip întreg `k`. În cazul în care numărul de ordine al valorii reale curent extrase `x` este un număr impar, atunci variabila `x` conţine cantitatea produsului cumpărat şi va fi salvată în variabila reală `c`, în caz contrar variabila `x` conţine preţul unitar şi va fi salvată în variabila `p`.

Totalul cheltuielilor presupune să calculăm mai întâi produsul dintre cantitate şi preţul unitar pentru fiecare produs cumpărat, apoi să calculam suma valorilor obţinute. Vom utiliza o variabilă

de tip real `total` pentru a reţine totalul cheltuielilor. Acesta este actualizată în momentul în care variabila `x` conţine preţul unitar prin adăugarea produsului dintre `p` şi `c`.

Extragerea unei valori reale dintr-un şir de caractere se poate realiza folosind funcţia `double atof (const char* str)`. Dacă întreg şirul nu poate fi convertit într-un număr real, atunci funcţia va returna numarul real obţinut din cel mai mare prefix al şirului care poate fi convertit într-un număr real sau valoarea 0 dacă nu exista un astfel de prefix. De exemplu, `atof("123.45")=123.45`, `atof("123.4abc56")=123.4`, iar `atof("abc123.4")=0`.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    FILE *fin = fopen("lista.txt", "r");
    char atom[50];
    double total=0, x, y,c,p;
    int k=0;

    while(fscanf(fin, "%s", atom) == 1)
    {
        x = atof(atom);
        if(x != 0)
        {
            k++;
            if(k%2 == 1)
                c = x;
            else
            {
                p = x;
                total += c*p;
            }
        }
    }

    printf("%.3lf",total);
    return 0;
}
```

5. Fişierul text *cuvinte.in* conţine pe prima linie două numere naturale a şi b ($a \leq b$), iar pe următoarele linii un text în care cuvintele sunt despărţite prin spaţii şi semnele de punctuaţie uzuale. Realizaţi un program care să scrie în fişierul text *cuvinte.out* toate cuvintele din fişierul cuvinte.in ale căror lungimi sunt cuprinse între a şi b sau mesajul "*Imposibil*" dacă în fişierul de intrare nu există nici un cuvânt cu proprietatea cerută.

Rezolvare:
În primul rând, vom citi de pe prima linie a fişierului text cele două numere naturale a şi b, folosind funcţia `fscanf`. După citirea celor două numere, în buffer-ul asociat fişierului va rămâne un caracter `'\n'`, pe care îl vom elimina printr-o citire în gol a unui caracter, folosind funcţia `fgetc`. Atenţie, dacă nu am elimina caracterul `'\n'` din buffer, prima linie care va fi citită din

fişier, folosind funcţia `fgets`, ar fi una formată doar din acel caracter `'\n'`, ci nu prima linie din fişier! În cazul problemei date acest aspect nu influenţează rezultatul, dar în cazul altor probleme se poate obţine un rezultat incorect (de exemplu, dacă am dori să calculăm şi câte linii conţine textul din fişier, am obţine o valoare mai mare cu 1 decât valoarea corectă). În continuare, vom citi linie cu linie conţinutul fişierului text folosind funcţia `fgets`, o vom împărţi în cuvinte folosind funcţia `strtok` şi vom scrie în fişierul de ieşire doar cuvintele ale căror lungimi se încadrează între *a* şi *b*.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    FILE *fin, *fout;
    int g, lmin, lmax;
    char linie[1001], *cuv;
    char delim[] = " .,:;?!\n";

    fin = fopen("cuvinte.in", "r");
    fout = fopen("cuvinte.out", "w");

    fscanf(fin, "%d %d", &lmin, &lmax);
    fgetc(fin);

    g = 0;
    while(fgets(linie, 1001, fin) != NULL)
    {
        cuv = strtok(linie, delim);
        while(cuv != NULL)
        {
            if(strlen(cuv) >= lmin && strlen(cuv) <= lmax)
            {
                fprintf(fout, "%s\n", cuv);
                g = 1;
            }
            cuv = strtok(NULL, delim);
        }
    }

    if(g == 0) fprintf(fout, "Imposibil!");

    fclose(fin);
    fclose(fout);
    return 0;
}
```

Variabila `g` este folosită pentru a putea verifica faptul că am găsit cel puţin un cuvânt cu proprietatea cerută sau nu.

6. Un algoritm de compresie presupune codificarea reversibilă a datelor astfel încât acestea să fie stocate într-o zonă memorie de dimensiune mai mică decât cea iniţială şi în urma procesului

de decompresie să se obţină exact datele iniţiale. Un algoritm simplu de compresie a unui şir de cifre constă în înlocuirea unei secvenţe de forma $\underbrace{x, x, x, \dots, x}_{y\ ori}$ cu perechea (x, y). Pe fiecare linie a fişierului text *comprimat.in* se află câte un şir de cifre codat folosind algoritmul de compresie descris anterior. Scrieţi un program care să realizeze decompresia datelor şi salvează rezultatele în fişierul *decomprimat.out*.

Exemplu:

comprimat.in	decomprimat.out
1,(2,3),1,4,(5,2)	1,2,2,2,1,4,5,5
2,3,4,5,6	2,3,4,5,6
(1,7),4,5	1,1,1,1,1,1,1,4,5
1,2,(3,4),5,6,(1,7),(2,3)	1,2,3,3,3,3,5,6,1,1,1,1,1,1,1,2,2,2

Rezolvare:

O metodă de rezolvare a acestei probleme presupune construirea unui nou şir de caractere pentru fiecare linie în parte. Astfel, fişierul este parcurs linie cu linie, folosind şirul de caractere `linie`. Noul şir `aux` va conţine, în ordinea apariţiei lor, toate cifrele de pe linia curentă `linie`, iar pentru fiecare pereche (x, y) din şirul `linie` va fi decomprimată, respectiv şirul `aux` va conţine cifra `x` de `y` ori. În acest sens, se propune aflarea adresei p unde apare caracterul '(' în şirul `linie` folosind funcţia `strchr`. Ulterior, se copiază în şirul `aux` toate caracterele din şirul `linie` aflate între adresa `linie` şi adresa `p`. Caracterul `x`, ce trebuie concatenat la şirul `aux` de `y` ori, se află la adresa `p+1`, iar caracterul `y` la adresa `p+3`. Deoarece funcţia `strcat` nu permite concatenarea unui şir de caractere cu un caracter vom construi un şir de caractere `wx`, de lungime 2, care să conţină pe prima poziţie caracterul `x`, iar pe a doua poziţie caracterul '`\0`'. Astfel, şirul `aux` este construit pas cu pas prin concatenarea de `y` ori a şirului `wx`, precum şi a caracterului virgulă. După decomprimarea unei perechi (x, y), se va elimina din şirul `linie` toate caracterele analizate, respectiv cele dintre adresa `linie` şi adresa `p+4`.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    FILE *fin, *fout;
    char linie[500], *p, wx[2], wy[2], *pp, aux[500];
    int t, i;

    fin = fopen("comprimat.txt","r");

    if(fin == NULL)
    {
        printf("Fisier inexistent!");
        return;
    }
```

```
    fout = fopen("decoprimat.txt","w");

    while(fgets(linie, 500, fin))
    {
        strcpy(aux, "");
        p = strchr(linie,'(');

        while(p != NULL)
        {
            strncat(aux, linie, p-linie);

            wx[0] = p[1];
            wx[1] = '\0';
            wy[0] = p[3];
            wy[1] = '\0';

            t = atoi(wy);

            for(i=0; i<t-1; i++)
            {
                strcat(aux,wx);
                strcat(aux,",");
            }

            strcat(aux,wx);
            strcpy(linie, p+5);
            p = strchr(linie,'(');
        }

        strcat(aux,linie);
        fprintf(fout, "%s\n", aux);

    }

    fclose(fin);
    fclose(fout);

    return 0;
}
```

7. Filtrarea automată a unui text este un proces prin care textul respectiv este mai întâi analizat în scopul de a detecta un anumit conţinut considerat inadecvat (cuvinte vulgare sau ofensatoare, distribuirea ilegală a unor fişiere multimedia, vânzarea de droguri sau arme de foc etc.), după care porţiunile de text considerate inadecvate sunt modificate sau chiar eliminate. Una dintre cele mai simple metode de filtrare a unui text constă în căutarea unor cuvinte dintr-o listă de cuvinte interzise şi înlocuirea lor cu un anumit caracter (de obicei, caracterul *). Scrieţi un program care să realizeze o astfel de filtrare a unui fişier text, folosind o listă de cuvinte interzise stocată în fişierul text *lista.txt*. Orice cuvânt interzis va fi format din cel mult 20 de litere şi sunt cel mult 100 de cuvinte interzise. Numele fişierului text de intrare se va citi de la tastatură, iar conţinutul său obţinut după operaţia de filtrare se va salva într-un fişier text cu acelaşi nume, dar prefixat cu *filtrat_* .

Rezolvare:
După ce vom citi de la tastatură numele fişierului text de intrare în variabila `nume_fin` şi vom construi numele fişierului text de ieşire în variabila `nume_fout`, vom încărca lista cuvintelor interzise din fişierul text *lista.txt* în tabloul de şiruri de caractere `lci`.

```
int i, j, nrci;
char lci[100][21], linie[1001], *p;
char nume_fin[101], nume_fout[101] = "filtrat_";
FILE *fin, *fout;

printf("Numele fisierului de intrare: ");
fgets(nume_fin, 101, stdin);
nume_fin[strlen(nume_fin)-1] = '\0';
strcat(nume_fout, nume_fin);

fin = fopen("lista.txt", "r");
nrci = 0;
while(fgets(lci[nrci], 21, fin) != NULL)
{
      if(lci[nrci][strlen(lci[nrci])-1] == '\n')
            lci[nrci][strlen(lci[nrci])-1] = '\0';
      nrci++;
}
fclose(fin);
```

După aceea, vom citi linie cu linie conţinutul fişierului text de intrare `fin` şi vom căuta în ea, pe rând, fiecare cuvânt interzis. Deoarece un cuvânt interzis poate să apară "mascat" şi în interiorul altui cuvânt, vom folosi funcţia `strstr` pentru a-l căuta, ci nu vom împărţi textul în cuvinte!

```
fin = fopen(nume_fin, "r");
fout = fopen(nume_fout, "w");

while(fgets(linie, 1001, fin) != NULL)
{
      for(i = 0; i < nrci; i++)
      {
            p = strstr(linie, lci[i]);
            while(p != NULL)
            {
                  for(j = p-linie; j < p - linie + strlen(lci[i]); j++)
                        linie[j] = '*';
                  p = strstr(p + strlen(lci[i]), lci[i]);
            }
      }

      fputs(linie, fout);
}

fclose(fin);
fclose(fout);
```

Cum ar trebui modificat programul de mai sus astfel încât să identifice în text exact cuvintele care sunt interzise? Dar astfel încât să identifice doar cuvintele care au ca prefix un cuvânt interzis?

8. Implementaţi o funcţie care să furnizeze toate cuvintele dintr-un şir de caractere formate din aceleaşi litere cu ale unui cuvânt dat (fără a fi neapărat anagrame!). Dacă şirul nu conţine nici un cuvânt cu proprietatea cerută, funcţia trebuie să returneze o valoare corespunzătoare. Cuvintele din şir sunt despărţite între ele prin spaţii şi semnele de punctuaţie uzuale. De exemplu, pentru şirul "Langa o cabana, stand pe o banca, un bacan a spus un banc bun." şi cuvântul "bacan" funcţia trebuie să furnizeze cuvintele "cabana", "banca", "bacan" şi "banc".

Rezolvare:
Pentru a verifica faptul că două cuvinte sunt formate din aceleaşi litere putem folosi funcţia `int strspn(const char* s, const char *t)` din biblioteca `string.h`, care returnează lungimea prefixului maximal al şirului `s` format doar din caractere ale şirului `t`. De exemplu, `strspn("program","ortoped")=3`, iar `strspn("program","test")=0`. Evident, dacă toate caracterele din şirul `s` se regăsesc în şirul `t`, atunci funcţia va returna chiar lungimea şirului `s`. Totuşi, este posibil ca şirul `t` să mai conţină şi alte caractere în afara celor ale şirului `s`, deci pentru a verifica faptul că şirurile `s` şi `t` sunt formate din aceleaşi caractere, trebuie să folosim condiţia `strspn(s,t)==strlen(s) && strspn(t,s)==strlen(t)`.

În continuare, vom nota prin `sir` şi `cuv` cele două date de intrare.

În prima variantă de rezolvare, vom construi în funcţie un şir de caractere `lc` alocat dinamic care va conţine toate cuvintele din şir cu proprietatea cerută, despărţite între ele prin câte un spaţiu. Dacă şirul dat `sir` nu conţine nici un cuvânt cu proprietatea cerută, atunci `lc` va rămâne un şir vid.

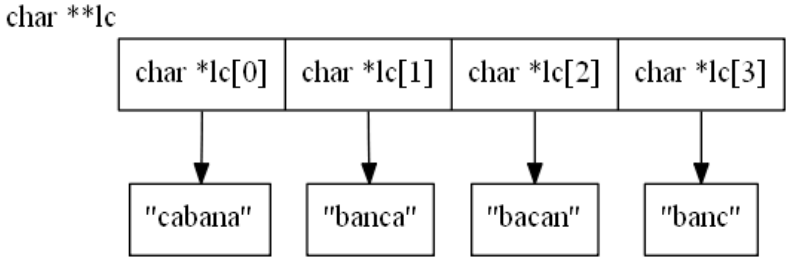
```
char* varianta1(char *sir, char *cuv)
{
    char *p, *lc, sep[] = " .,:;!?\n";

    lc = (char *)calloc(strlen(sir) + 1, sizeof(char));

    p = strtok(sir, sep);
    while(p != NULL)
    {
        if(strspn(p, cuv) == strlen(p) && strspn(cuv, p) == strlen(cuv))
        {
            strcat(lc, p);
            strcat(lc, " ");
        }
        p = strtok(NULL, sep);
    }

    if(strlen(lc) > 0)
        lc[strlen(lc)-1] = '\0';

    lc = (char *)realloc(lc, strlen(lc) + 1);
```

```
    return lc;
}
```

Deoarece lungimea şirului rezultat `lc` nu poate depăşi lungimea şirului dat `sir`, am alocat dinamic şirul `lc` pe un număr de octeţi egal cu numărul de octeţi pe care este memorat `sir` şi l-am iniţializat direct cu un şir vid, folosind funcţia `calloc`. După ce am adăugat în şirul `lc` toate cuvintele cu proprietatea cerută, am optimizat spaţiul de memorie utilizat pentru el, redimensionându-l pe un număr de octeţi egal cu lungimea sa fizică. Evident, o altă modalitate de alocare dinamică a şirului `lc` constă în iniţializarea sa cu un şir vid, urmată de redimensionarea sa înaintea fiecărei concatenări cu un cuvânt `p` având proprietatea cerută:

```
char* varianta1(char *sir, char *cuv)
{
    char *p, *lc, sep[] = " .,:;!?\n";

    lc = (char *)calloc(1, sizeof(char));
    p = strtok(sir, sep);
    while(p != NULL)
    {
        if(strspn(p, cuv) == strlen(p) && strspn(cuv, p) == strlen(cuv))
        {
            lc = (char *)realloc(lc, strlen(lc)+strlen(p)+2);
            strcat(lc, p);
            strcat(lc, " ");
        }

        p = strtok(NULL, sep);
    }

    if(strlen(lc) > 0)
        lc[strlen(lc)-1] = '\0';

    return lc;
}
```

Metoda prezentată mai sus are un dezavantaj, respectiv cuvintele returnate de funcţie nu pot fi accesate individual în mod direct, deoarece sunt concatenate într-un singur şir. Pentru a elimina acest dezavantaj, în a doua variantă de rezolvare vom construi în funcţie un tablou de şiruri de caractere alocat dinamic, având următoarea structură:



Deoarece tabloul de şiruri va fi alocat dinamic în funcţie, parametrul corespunzător `lc` de tip pointer dublu trebuie transmis prin adresă. Practic, parametrul `lc` va conţine adresa primului element al tabloului de şiruri, iar această adresă va fi stabilită în funcţie, în momentul alocării tabloului de şiruri. Evident, funcţia trebuie să furnizeze şi numărul de elemente ale tabloului `lc`, prin intermediul unui parametru `nrc` transmis tot prin adresă.

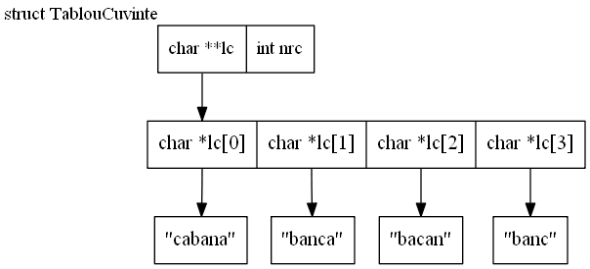
```
void varianta2(char *sir, char *cuv, char ***lc, int *nrc)
{
    char *p, sep[] = " .,:;!?\n";

    *nrc = 0;
    *lc = NULL;
    p = strtok(sir, sep);
    while(p != NULL)
    {
        if(strspn(p, cuv) == strlen(p) && strspn(cuv, p) == strlen(cuv))
        {
            *lc = (char **)realloc(*lc, (*nrc+1)*sizeof(char*));
            (*lc)[*nrc] = (char *)malloc(strlen(p) + 1);
            strcpy((*lc)[*nrc], p);
            (*nrc)++;
        }

        p = strtok(NULL, sep);
    }
}
```

După apelarea funcţiei, trebuie eliberat tot spaţiul de memorie utilizat:

```
char **lc;
int i, nrc;

varianta2(prop2, cuv, &lc, &nrc);

if(lc == NULL)
     printf("Imposibil");
else
{
     printf("Cuvintele avand aceleasi litere cu %s:\n", cuv);
     for(i = 0; i < nrc; i++)
        printf("%s\n", lc[i]);

     for(i = 0; i < nrc; i++)
        free(lc[i]);
     free(lc);
}
```

Deoarece parametrii `lc` şi `nrc` sunt transmişi prin adrese, se observă faptul că este necesară dereferenţierea lor repetată în corpul funcţiei. Astfel, luând în considerare şi aspecte referitoare la priorităţile operatorilor, am obţinut un cod complicat, greu de urmărit. Acest dezavantaj poate

fi eliminat prin definirea unei structuri `TablouCuvinte` care să înglobeze atât tabloul de şiruri, cât şi numărul său de elemente:



Astfel, funcţia va returna o valoare de tip `TablouCuvinte`, deci parametrii transmişi prin adrese `lc` şi `nrc` nu mai sunt necesari.

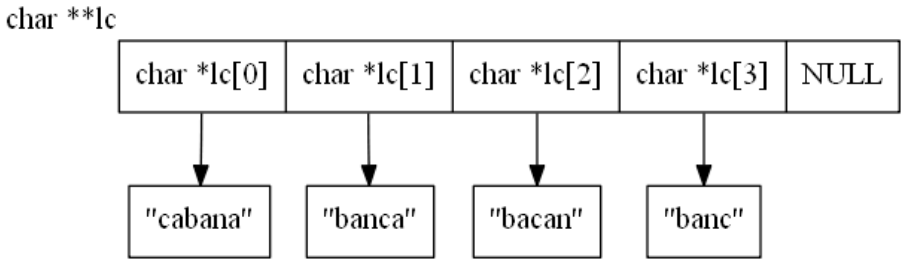
```
typedef struct
{
    int nrc;
    char **lc;
}TablouCuvinte;

TablouCuvinte varianta3(char *sir, char *cuv)
{
    TablouCuvinte tc;
    char *p, sep[] = " .,:;!?\n";

    tc.nrc = 0;
    tc.lc = NULL;
    p = strtok(sir, sep);
    while(p != NULL)
    {
        if(strspn(p, cuv) == strlen(p) && strspn(cuv, p) == strlen(cuv))
        {
            tc.lc = (char **)realloc(tc.lc, (tc.nrc+1)*sizeof(char*));
            tc.lc[tc.nrc] = (char *)malloc(strlen(p) + 1);
            strcpy(tc.lc[tc.nrc], p);
            tc.nrc++;
        }

        p = strtok(NULL, sep);
    }

    return tc;
}
```

După apelarea funcţiei, trebuie eliberat tot spaţiul de memorie utilizat:

```
TablouCuvinte tc;
int i;

tc = varianta3(prop4, cuv);
```

```
if(tc.lc == NULL)
      printf("Imposibil");
else
{
      printf("Cuvintele avand aceleasi litere cu %s:\n", cuv);
      for(i = 0; i < tc.nrc; i++)
         printf("%s\n", tc.lc[i]);

      for(i = 0; i < nrc; i++)
         free(tc.lc[i]);
      free(tc.lc);
}
```

O altă modalitate prin care pot fi eliminaţi cei doi parametrii `lc` şi `nrc` transmişi prin adrese o reprezintă adăugarea unui pointer NULL după ultimul element al tabloului `lc`:



Astfel, funcţia va returna doar adresa de început a tabloului `lc`:

```
char** varianta3(char *sir, char *cuv)
{
    char *p, sep[] = " .,:;!?\n";
    char **lc;
    int nrc;

    nrc = 0;
    lc = NULL;
    p = strtok(sir, sep);
    while(p != NULL)
    {
        if(strspn(p, cuv) == strlen(p) && strspn(cuv, p) == strlen(cuv))
        {
            lc = (char **)realloc(lc, (nrc+1)*sizeof(char*));
            lc[nrc] = (char *)malloc(strlen(p) + 1);
            strcpy(lc[nrc], p);
            nrc++;
        }

        p = strtok(NULL, sep);
    }

    if(lc != NULL)
    {
        lc = (char **)realloc(lc, (nrc+1)*sizeof(char*));
```

```
        lc[nrc] = NULL;
    }

    return lc;
}
```

După apelarea funcţiei, trebuie eliberat tot spaţiul de memorie utilizat:

```
char **lc, **p;

lc = varianta3(prop3, cuv);

if(lc == NULL)
      printf("Imposibil");
else
{
      printf("Cuvintele avand aceleasi litere cu %s:\n", cuv);
      for(p = lc; *p != NULL; p++)
         printf("%s\n", *p);

      for(p = lc; *p != NULL; p++)
         free(*p);
      free(lc);
}
```

O modalitate echivalentă de parcurgere a tabloului `lc` este următoarea:

```
for(i = 0; lc[i] != NULL; i++)
   printf("%s\n", lc[i]);
```

În general, problema poate fi privită ca una de parsare (analiză lexicală) a unui şir de caractere, prin care sunt extrase cuvintele cu o anumită proprietate.

9. Scrieţi un program care să afişeze pe ecran cuvintele distincte dintr-un fişier text în ordinea descrescătoare a frecvenţelor lor de apariţie, iar în cazul unor frecvenţe egale, cuvintele vor fi afişate în ordine alfabetică. Nu se va face distincţie între litere mici şi litere mari. Textul poate fi împărţit pe mai multe linii, iar pe o linie cuvintele sunt despărţite între ele prin semnele de punctuaţie uzuale. Orice linie din fişier conţine cel mult 500 de caractere. Fişierul text conţine cel mult 1000 de cuvinte, fiecare cuvânt fiind format din cel mult 20 de caractere. Numele fişierului text se va citi de la tastatură.

Rezolvare:

În limbajul C standard nu există nicio funcţie care să transforme toate literele mari dintr-un şir de caractere în litere mici (sau invers). Totuşi, în biblioteca `ctype.h` există funcţiile `int tolower(int c)` şi `int toupper(int c)` care returnează litera mică/litera mare corespunzătoare literei mari/mici `c` (dacă `c` nu este o literă mare/mică, atunci ambele funcţii vor returna chiar valoarea `c`). Folosind funcţia `tolower` vom scrie, mai întâi, o funcţie care să furnizeze şirul obţinut prin înlocuirea tuturor literelor mari dintr-un şir de caractere dat *s* în litere mici:

```
char *toLower(char *s)
{
    int i;
    char *aux;

    aux = (char *)malloc(strlen(s)+1);
    for(i = 0; i < strlen(s); i++)
        aux[i] = tolower(s[i]);
    aux[i] = '\0';

    return aux;
}
```

Pentru a calcula frecvenţele de apariţie ale cuvintelor distincte din text, vom folosi o structură care va conţine un cuvânt şi frecvenţa sa de apariţie, precum şi un tablou unidimensional *t* cu elemente de acest tip. După ce vom citi câte o linie din fişierul text, o vom împărţi în cuvinte şi apoi vom căuta cuvântul curent în tabloul *t*. Dacă acesta se găseşte deja în tabloul *t*, îi vom creşte frecvenţa, altfel îl vom adăuga la sfârşitul tabloului *t*, cu frecvenţa 1.
Pentru sortarea elementelor tabloului *t* în ordinea cerută vom folosi funcţia `qsort` din biblioteca `stdlib.h` şi funcţia comparator `cmpCuvinte`, definită mai jos.

```
typedef struct
{
    char cuv[21];
    int fr;
} Cuvant;

int cmpCuvinte(const void *a, const void *b)
{
    Cuvant va = *(Cuvant*)a;
    Cuvant vb = *(Cuvant*)b;

    if(va.fr < vb.fr)
        return 1;
    if(va.fr > vb.fr)
        return -1;
    return strcmp(va.cuv, vb.cuv);
}

int main()
{
    char linie[501], numef[51], *p, sep[] = " .,:;!?\n";
    FILE *f;
    Cuvant t[1000];
    int i, nrc;

    printf("Numele fisierului text: ");
    fgets(numef, 51, stdin);
    numef[strlen(numef)-1] = '\0';

    f = fopen(numef, "r");
```

```
    nrc = 0;
    while(fgets(linie, 501, f) != NULL)
    {
        p = strtok(linie, sep);
        while(p != NULL)
        {
            for(i = 0; i < nrc; i++)
                if(strcmp(toLower(p), t[i].cuv) == 0)
                    break;

            if(i < nrc)
                t[i].fr++;
            else
            {
                strcpy(t[i].cuv, toLower(p));
                t[i].fr = 1;
                nrc++;
            }

            p = strtok(NULL, sep);
        }
    }

    fclose(f);

    qsort(t, nrc, sizeof(Cuvant), cmpCuvinte);

    printf("Cuvintele distincte din fisier si frecventele lor:\n");
    for(i = 0; i < nrc; i++)
        printf("%s -> %d\n", t[i].cuv, t[i].fr);

    return 0;
}
```

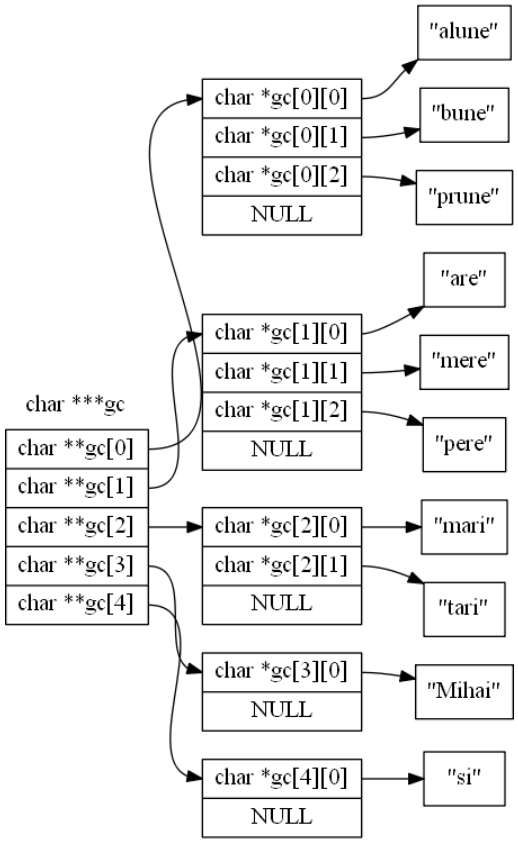
10. Fişierul text *cuvinte*.*in* conţine un text care poate fi împărţit pe mai multe linii, iar pe o linie cuvintele sunt despărţite între ele prin semnele de punctuaţie uzuale. Realizaţi un program care să scrie în fişierul text *cuvinte*.*out* grupurile de cuvinte din fişierul *cuvinte*.*in* care rimează între ele. Două cuvinte rimează dacă ultimele lor două litere sunt egale. Grupurile vor fi scrise câte unul pe linie, în ordinea descrescătoare a numărului de cuvinte pe care le conţin, iar în fiecare grup cuvintele vor fi ordonate alfabetic.

Exemplu:

cuvinte.in	**cuvinte.out**
Mihai are prune mari, mere bune, pere tari si alune.	alune bune prune are mere pere mari tari Mihai si

Rezolvare:

O metodă de rezolvare a acestei probleme constă în crearea unui grup de cuvinte pentru fiecare rimă distinctă găsită, adică a unui tablou având elemente de tip şir de caractere. Grupurile obţinute trebuie memorate într-un tablou, deci vom avea nevoie de "un tablou de tablouri de şiruri de caractere", adică de un tablou tridimensional de caractere. Deoarece nu cunoaştem şi nici nu putem estima uşor dimensiunile acestui tablou (prima dimensiune ar reprezenta numărul maxim de grupuri de cuvinte care se pot forma, a doua dimensiune ar reprezenta numărul maxim de cuvinte dintr-un grup, iar a treia dimensiune ar indica lungimea maximă a unui cuvânt), vom folosi o structură de date gc alocată dinamic prin intermediul unui pointer triplu. Pentru exemplul de mai sus, tabloul gc va avea structura din figura alăturată. Numărul curent de elemente din tabloul gc (adică numărul curent de grupuri de cuvinte care rimează) va fi memorat în variabila nrg.



Pentru a verifica faptul că două cuvinte rimează sau nu vom folosi următoarea funcţie:

```
int rimeaza(const char *s, const char *t)
{
    if(s == NULL || t == NULL) return 0;

    if(strlen(s) < 2 || strlen(t) < 2) return 0;

    return strcmp(s+strlen(s)-2, t+strlen(t)-2) == 0;
}
```

Pentru a nu adăuga de mai multe ori un cuvânt într-un grup, vom folosi următoarea funcţie de căutare:

```
int cauta(char **grup, char *cuvant)
{
    char **p;

    for(p = grup; *p != NULL; p++)
        if(strcmp(*p, cuvant) == 0)
            return 1;
    return 0;
}
```

Deoarece nu am păstrat într-o structură de date adecvată numărul de cuvinte din fiecare grup, avem nevoie de o funcţie care să determine numărul de cuvinte dintr-un grup:

```
int numara(char **grup)
{
    char **p;
    int nrc;

    nrc = 0;
    for(p = grup; *p != NULL; p++)
        nrc++;

    return nrc;
}
```

Pentru rezolvarea problemei vom parcurge, mai întâi, următorii paşi:

- vom citi câte o linie din fişierul de intrare `fin` în şirul de caractere `linie` şi o vom împărţi în cuvinte, folosind funcţia `strtok`;
- vom testa dacă există un grup de cuvinte `gc[i]` în care poate fi adăugat cuvântul curent `cuv`, verificând dacă acesta rimează cu primul cuvânt `gc[i][0]` din fiecare grup:
  - dacă există un astfel de grup `gc[i]`, vom redimensiona corespunzător grupul şi vom adăuga cuvântul `cuv` la sfârşit;
  - dacă nu există niciun astfel de grup, vom redimensiona corespunzător tabloul `gc` (adică vom adăuga un nou grup de cuvinte) şi vom adăuga cuvântul `cuv` ca prim element.

```
FILE *fin, *fout;
int nrg, i, nc;
char ***gc, linie[1001], *cuv, **p;
char delim[] = " .,:;?!\n";

fin = fopen("cuvinte.in", "r");

nrg = 0;
gc = NULL;
while(fgets(linie, 1001, fin) != NULL)
{
      cuv = strtok(linie, delim);
      while(cuv != NULL)
      {
            for(i = 0; i < nrg; i++)
            if(rimeaza(gc[i][0], cuv) == 1)
            {
                  if(cauta(gc[i], cuv) == 0)
                  {
                        nc = numara(gc[i]);
                        gc[i] = (char**)realloc(gc[i],(nc+2)*sizeof(char *));
                        gc[i][nc] = (char *)malloc(strlen(cuv) + 1);
                        strcpy(gc[i][nc], cuv);
                        gc[i][nc+1] = NULL;
                  }
                  break;
```

```
            }

            if(i == nrg)
            {
                  gc = (char ***)realloc(gc, (nrg+1)*sizeof(char **));
                  gc[nrg] = (char **)malloc(2*sizeof(char **));
                  gc[nrg][0] = (char *)malloc(strlen(cuv) + 1);
                  strcpy(gc[nrg][0], cuv);
                  gc[nrg][1] = NULL;
                  nrg++;
            }

            cuv = strtok(NULL, delim);
      }
}

fclose(fin);
```

Pentru a sorta alfabetic cuvintele din fiecare grup, vom folosi funcţia `qsort` din biblioteca `stdlib.h` şi funcţia comparator `cmpCuvinteGrup` definită astfel:

```
int cmpCuvinteGrup(const void *a, const void *b)
{
      const char *pa = *(const char **)a;
      const char *pb = *(const char **)b;

      return strcmp(pa, pb);
}

for(i = 0; i < nrg; i++)
      qsort(gc[i], numara(gc[i]), sizeof(char *), cmpCuvinteGrup);
```

Pentru a sorta grupurile de cuvinte în ordine descrescătoare după numărul de cuvinte conţinute, vom folosi funcţia `qsort` şi funcţia comparator `cmpCuvinteGrup` definită astfel:

```
int cmpGrupuriCuvinte(const void *a, const void *b)
{
      char **pa = *(char ***)a;
      char **pb = *(char ***)b;

      int ncga = numara(pa);
      int ncgb = numara(pb);

      if(ncgb != ncga)
            return ncgb - ncga;
      else
            return strcmp(*pa, *pb);
}

qsort(gc, nrg, sizeof(char **), cmpGrupuriCuvinte);
```

Se observă faptul că pot exista două sau mai multe grupuri cu acelaşi număr de cuvinte, caz în care am propus sortarea lor alfabetică în funcţie de primul cuvânt din fiecare grup.

După ce am sortat atât cuvintele din fiecare grup, cât şi grupurile, vom scrie rezultatul în fişierul text *cuvinte.out*:

```
fout = fopen("cuvinte.out", "w");

for(i = 0; i < nrg; i++)
{
      for(p = gc[i]; *p != NULL; p++)
            fprintf(fout, "%s ", *p);
      fprintf(fout, "\n");
}

fclose(fout);
```

La sfârşitul programului, vom elibera toate zonele de memorie alocate dinamic:

```
for(i = 0; i < nrg; i++)
{
      for(p = gc[i]; *p != NULL; p++)
            free(*p);
      free(gc[i]);
}

free(gc);
```

Soluţia prezentată nu este cea mai eficientă din punct de vedere al timpului de executare. O soluţie mai eficientă ar presupune utilizarea unor structuri de date complexe, de tipul tabelelor de dispersie.

În general, problema poate fi privită ca una de parsare (analiză lexicală) a unui şir de caractere, prin care sunt extrase şi apoi împărţite în grupuri cuvintele cu o anumită proprietate (clasificare).

Probleme propuse

1. Scrieţi o funcţie care să verifice dacă un şir de caractere t se poate obţine concatenând un şir s cu el însuşi de un număr arbitrar de ori sau nu. În caz afirmativ, funcţia va returna numărul de concatenări necesare, altfel va întoarce valoarea -1.

2. Fişierul $cuvinte.in$ conţine pe prima linie un cuvânt c format din n ≥ 2 litere mici din alfabetul englez şi un număr natural nenul p ($2 \leq p \leq n$), iar pe următoarele linii un text în care cuvintele sunt despărţite prin spaţii şi semnele de punctuaţie uzuale. Realizaţi un program care să scrie în fişierul text $cuvinte.out$ toate cuvintele din fişierul $cuvinte.in$ care sunt p*-rime* ale cuvântului c sau mesajul "Imposibil" dacă în fişierul de intrare nu există nici un cuvânt cu proprietatea cerută. Două cuvinte sunt p*-rime* dacă au ultimele p caractere identice.

3. O metodă simplă (dar nesigură!!!) de criptare a unui text o reprezintă *substituţia monoalfabetică*, prin care o literă x dintr-un text dat este înlocuită printr-o altă literă y, întotdeauna aceeaşi. Scrieţi un program care să realizeze criptarea, respectiv decriptarea, unui fişier text al cărui nume se citeşte de la tastatură. Regula de substituţie pentru cele 26 de litere mici ale alfabetului limbii engleze este dată în fişierul text *cheie.txt*, fiecare linie conţinând câte o pereche de litere x şi y cu semnificaţia de mai sus.

4. Fişierul $cuvinte.in$ conţine pe prima linie un cuvânt w format din $n \geq 1$ litere mici din alfabetul englez, iar pe următoarele linii un text în care cuvintele sunt despărţite prin spaţii şi semnele de punctuaţie uzuale. Realizaţi un program care să scrie în fişierul text $cuvinte.out$ toate poziţiile unde apare cuvântul w în fişierul $cuvinte.in$ sau mesajul "Imposibil" dacă în fişierul de intrare nu apare deloc cuvântul w. O poziţie cuvântului w în textul dat se va indica sub forma unei perechi (l, c), unde l reprezintă numărul de ordine al liniei pe care apare cuvântul w, iar c indică al câtelea *cuvânt* este w pe linia l. Atenţie, cuvântul w poate să apară de mai multe ori pe aceeaşi linie!

5. Fişierul text $dictionar.txt$ conţine pe prima linie un număr natural nenul n, iar pe fiecare din următoarele n linii câte o pereche de cuvinte, primul fiind în limba română, iar al doilea în limba engleză.

Un cuvânt din limba română format din n litere poate fi considerat, în raport cu limba engleză, ca fiind:

1. *traductibil* – dacă el se găseşte exact în forma dată în dicţionar;
2. *parţial traductibil* – dacă el nu este traductibil, dar un prefix al său de lungime $n-1$ sau $n-2$ se găseşte în dicţionar;
3. *intraductibil* – dacă el nu este nici traductibil şi nici parţial traductibil.

Pentru un text în limba română se defineşte indicele de traductibilitate ca fiind raportul dintre numărul de cuvinte traductibile sau parţial traductibile şi numărul total de cuvinte din text. Scrieţi un program care să calculeze indicele de traductibilitate în limba engleză al unui text în limba română, memorat în fişierul text $exemplu.txt$.

Exemplu

dictionar_engleza.txt
12 inaltime altitude caine dog muncitor worker zi day pisica cat lucra work bloc block cateva few pentru for noapte night un a inalt high
exemplu.txt
muncitorii au lucrat cateva zile pentru a ridica un bloc inalt

În textul în limba română sunt:

- 5/3/3 cuvinte traductibile/parțial traductibile/intraductibile în limba engleză, deci indicele său de traductibilitate este (5+3)/(5+3+3) = 8/11 = 0.73