9 Şiruri de caractere.

9.1. Generalităţi.

O constantă şir de caractere se reprezintă intern printr-un tablou de caractere terminat prin caracterul nul **'\0'**, memoria alocată fiind lungimea şirului + 1 (1 octet se adaugă pentru caracterul terminator al şirului).

Un tablou de caractere poate fi iniţializat fără a-i specifica dimensiunea:

```
char salut[]={'B','o','n','j','o','u','r','!','\0'};
```

sau mai simplu, specificând şirul între ghilimele:

```
char salut[]="Bonjour!"
```

(tabloul este iniţializat cu conţinutul şirului de caractere -se alocă 9 octeţi)

Folosirea unui pointer la un şir de caractere iniţializat nu copiază şirul, ci are următorul efect:

- se alocă memorie pentru şirul de caractere, inclusiv terminatorul nul la o adresă fixă de memorie
- se iniţializează spaţiul cu valorile constantelor caractere
- se iniţializează pointerul **Psalut** cu adresa spaţiului alocat

```
char *Psalut="Buna ziua!";
```

(pointerul este iniţializat să indice o constantă şir de caractere)

Aşadar, în C *nu există operaţia de atribuire de şiruri de caractere* (sau în general de atribuire de tablouri), ci numai atribuire de pointeri - atribuirea **t=s** nu copiază un tablou, pentru aceasta se foloseşte funcţia **strcpy(t, s)**.

Pentru a uşura lucrul cu şiruri de caractere, în biblioteca standard sunt prevăzute o serie de funcţii, ale căror prototipuri sunt date în fişierele **<ctype.h>** şi **<string.h>**.

În fişierul **<ctype.h>** există funcţii (codificate ca macroinstrucţiuni) care primesc un parametru întreg, care se converteşte în **unsigned char**, şi întorc rezultatul diferit de 0 sau egal cu 0, după cum caracterul argument satisface sau nu condiţia specificată:

```
islower(c)  1 dacă c∈ {'a'..'z'}
isupper(c)  1 dacă c∈ {'A'..'Z'}
isalpha(c)  1 dacă c∈ {'A'..'Z'}∨{'a'..'z'}
isdigit(c)  1 dacă c∈ {'0'..'9'}
isxdigit(c) 1 dacă c∈ {'0'..'9'}∨{'A'..'F'}∨{a'..'f'}
isalnum(c)  1 dacă isalpha(c)||isdigit(c)
isspace(c)  1 dacă c∈ {' ','\n','\t','\r','\f','\v'}
isgraph(c)  1 dacă c este afişabil, fără spaţiu
isprint(c)  1 dacă c este afişabil, cu spaţiu
iscntrl(c)  1 dacă c este caracter de control
ispunct(c)  1 dacă isgraph(c) && !isalnum(c)
```

Conversia din literă mare în literă mică şi invers se face folosind funcţiile: **tolower(c)** şi **toupper(c).**

Exemplul 19*: Scrieţi o funcţie care converteşte un şir de caractere reprezentând un număr întreg, într-o valoare întreagă. Numărul poate avea semn şi poate fi precedat de spaţii albe.*

```
#include <ctype.h>
int atoi(char *s)
{ int i, nr, semn;
  for(i=0; isspace(s[i]); i++) /*ignora spatii albe*/
              ;
  semn=(s[i]=='-')?-1:1;       /*stabilire semn*/
  if(s[i]=='+'||s[i]=='-')     /*se sare semnul*/
    i++;
  for(nr=0;isdigit(s[i]);i++)  /*conversie in cifra*/
    nr=10*nr+(s[i]-'0');       /*si alipire la numar*/
  return semn*nr;
}
```

Exemplul 20: ***Scrieţi o funcţie care converteşte un întreg într-un şir de caractere în baza 10.***

Algoritmul cuprinde următorii paşi:

```
{ se extrage semnul numărului;
  se extrag cifrele numărului, începând cu cmps;
  se transformă în caractere şi se depun într-un tablou;
  se adaugă semnul şi terminatorul de şir;
  se inversează şirul;
}

void inversare(char[]);
void itoa(int n, char s[]){
  int j, semn;
  if((semn=n)<0)
    n=-n;
  j=0;
  do
    s[j++]=n%10+'0';
  while ((n/=10)>0);
  if(semn<0)
    s[j++]='-';
  s[j]='\0';
  inversare(s);
}
void inversare(char s[])
{ int i,j;
  char c;
  for(i=0,j=strlen(s)-1;i<j;i++,j--)
    c=s[i], s[i]=s[j], s[j]=c;
}
```

Exemplul 21*: Scrieţi o funcţie care converteşte un întreg fără semn într-un şir de caractere în baza 16.*

Pentru a trece cu uşurinţă de la valorile cifrelor hexazecimale `0,1,…15` la caracterele corespunzătoare; `'0','1',…,'a',…,'f',` vom utiliza un tablou iniţializat de caractere.

```
static char hexa[]="0123456789abcdef";
void itoh(int n, char s[])
```

```
{ j=0;
  do {
      s[j++]=hexa[n%16];
  while ((n/=16)>0);
  s[j]='\0';
  inversare(s);
}
```

Fişierul **<string.h>** conţine prototipurile următoarelor funcţii:

Tabel 9.1. Semnăturile funcţiilor din fişierul antet string.h

`char* strcpy(char* d,const char* s)`	copiază şirul **s** în **d**, inclusiv **'\0'**, întoarce **d**
`char* strncpy(char* d, const char* s, int n)`	copiază **n** caractere din şirul **s** în **d**, completând eventual cu **'\0'**, întoarce **d**
`char* strcat(char* d,const char* s)`	adaugă şirul **s** la sfârşitul lui **d**, întoarce **d**
`char* strncat(char* d,const char* s, int n)`	concatenează cel mult **n** caractere din şirul **s** la sfârşitul lui **d**, completând cu **'\0'**, întoarce **d**
`int strcmp(const char* d, const char* s)`	compară şirurile **d** şi **s**, întoarce **-1** dacă **d<s**, **0** dacă **d==s** şi **1** dacă **d>s**
`int stricmp(const char* d, const char* s)`	compară şirurile **d** şi **s** (ca şi **strcmp()**) fără a face distincţie între litere mari şi mici
`int strncmp(const char* d, const char* s, int n )`	similar cu **strcmp()**, cu deosebirea că se compară cel mult **n** caractere
`int strincmp(const char* d, const char* s, int n )`	similar cu **strncmp()**,cu deosebirea că nu se face distincţie între literele mari şi mici
`char* strchr(const char* d,char c)`	caută caracterul **c** în şirul **d;** întoarce un pointer la prima apariţie a lui **c** în **d,** sau **NULL**
`char* strrchr(const char* d,char c)`	întoarce un pointer la ultima apariţie a lui **c** în **d,** sau **NULL**
`char* strstr(const char* d, const char* s)`	întoarce un pointer la prima apariţie a subşirului **s** în **d,** sau **NULL**
`char* strpbrk(const char* d, const char* s)`	întoarce un pointer la prima apariţie a unui caracter din subşirul **s** în **d,** sau **NULL**
`int strspn(const char* d, const char* s)`	întoarce lungimea prefixului din **d** care conţine numai caractere din **s**
`int strcspn(const char* d, const char* s)`	întoarce lungimea prefixului din **d** care conţine numai caractere ce nu apar în **s**
`int strlen(const char* s)`	întoarce lungimea lui **s** (**'\0'** nu se numără)
`char* strlwr(char* s)`	converteşte literele mari în litere mici în **s**
`char* strupr(char* s)`	converteşte literele mici în litere mari în **s**

`char* strtok(const char* d, const char* s)`	caută în **d** subşirurile delimitate de caracterele din **s**;primul apel întoarce un pointer la primul subşir din **d** care nu conţine caractere din **s** următoarele apeluri se fac cu primul argument **NULL**, întorcându-se de fiecare dată un pointer la următorul subşir din **d** ce nu conţine caractere din **s**; în momentul în care nu mai există subşiruri, funcţia întoarce **NULL**

Ca exerciţiu, vom codifica unele din funcţiile a căror prototipuri se găsesc în **<string.h>**, scriindu-le în două variante: cu tablouri şi cu pointeri.

Exemplul 22:*Scrieţi o funcţie având ca parametru un şir de caractere, care întoarce lungimea sirului*

```
/*varianta cu tablouri*/
int strlen(char *s)
{ int j;
  for(j=0; s[j]; j++)
          ;
  return j;
}

/*varianta cu pointeri*/
int strlen(char *s)
{ char *p=s;
  while(*p)
    p++;
  return p-s;
}
```

Exemplul 23: *Scrieţi o funcţie care copiază un şir de caractere* **s** *în* **d**.

```
/*varianta cu tablouri*/
void strcpy(char *d, char *s)
{ int j=0;
  while((d[j]=s[j])!='\0')
    j++;
}

/*varianta cu pointeri*/
void strcpy(char *d, char *s)
{ while((*d=*s)!='\0'){
    d++;
    s++;
}
```

Postincrementarea pointerilor poate fi făcută în operaţia de atribuire deci:

```
void strcpy(char *d, char *s){
  while((*d++=*s++)!='\0')
```

```
            ;
}
```

Testul faţă de `'\0'` din **while** este redundant, deci putem scrie:

```
void strcpy(char *d, char *s){
  while(*d++=*s++)
            ;
}
```

Exemplul 24: *Scrieţi o funcţie care compară lexicografic două şiruri de caractere şi întoarce rezultatul* **−1**, **0** *sau* **1** *după cum* **d<s**, **d==s** *sau* **d>s.**

```
/*varianta cu tablouri*/
int strcmp(char *d, char *s)
{ int j;
  for(j=0; d[j]==s[j]; j++)
    if(d[j]=='\0')
      return 0;
  return d[j]-s[j];
}

/*varianta cu pointeri*/
int strcmp(char *d, char *s)
{ for(; *d==*s; d++,s++)
    if(*d=='\0')
      return 0;
  return *d-*s;
}
```

Exemplul 25: *Scrieţi o funcţie care determină poziţia (indexul) primei apariţii a unui subşir* **s** *într-un şir* **d**. *Dacă* **s** *nu apare în* **d**, *funcţia întoarce* **−1.**

```
int strind(char d[], char s[]){
   int i,j,k;
   for (i=0; d[i]; i++) {
      for (j=i,k=0; s[k] && d[j]==s[k]; j++,k++)
                     ;
      if (k>0 && s[k]=='\0')
         return i;
   }
   return -1;
}
```

9.2. Funcţii de intrare / ieşire relative la şiruri de caractere.

Pentru a citi un şir de caractere de la intrarea standard se foloseşte funcţia **gets()** având prototipul:

```
char *gets(char *s);
```

Funcţia **gets()** citeşte caractere din fluxul standard de intrare **stdin** în zona de memorie adresată de pointerul **s**. Citirea continuă până la întâlnirea sfârşitului de linie. Marcajul de sfârşit de linie nu este copiat, în locul lui fiind pus caracterul nul (**'\0'**). Funcţia întoarce adresa zonei de memorie în

care se face citirea (adică **s**) sau **NULL**, dacă în locul şirului de caractere a fost introdus marcajul de sfârşit de fişier.
Pentru a scrie un şir de caractere terminat prin caracterul **NULL**, la ieşirea standard **stdout**, se foloseşte funcţia:

```
int puts(char *s);
```

Caracterul terminator nu este transmis la ieşire, în locul lui punându-se marcajul de sfârşit de linie.
Caracterele citite într-un tablou ca un şir de caractere (cu **gets()**) pot fi convertite sub controlul unui format folosind funcţia:

```
int sscanf(char *sir, char *format, adrese_var_formatate);
```

Singura deosebire faţă de funcţia **scanf()** constă în faptul că datele sunt preluate dintr-o zonă de memorie, adresată de primul parametru (şi nu de la intrarea standard).

Exemplul 26: ***Scrieţi o funcţie care citeşte cel mult n numere reale, pe care le plasează într-un tablou x. Funcţia întoarce numărul de valori citite.***

Vom citi numerele într-un şir de caractere **s**. De aici vom extrage în mod repetat câte un număr, folosind funcţia **sscanf()** şi îl vom converti folosind un format corespunzător. Ciclul se va repeta de **n** ori, sau se va opri când se constată că s-au terminat numerele.
Vom scrie funcţia în 2 variante: folosind tablouri sau folosind pointeri.

```
/* varianta cu tablouri */
int citreal(int n, double x[])
{ char s[255];
  int j;
  double y;
  for (j=0; j<n; j++) {
    if(gets(s)==NULL)
      return j;
    if(sscanf(s,"%lf",&y)!=1) /*conversie in real*/
      break;              /*s-au terminat numerele*/
  x[j]=y;
  }
  return j;
}

/* varianta cu pointeri */
int citreal(int n, double *px)
{ int j=0;
  double y;
  double *p=px+n;
  while(px<p) {
    if(gets(s)==NULL)
      return j;
    if(sscanf(s,"%lf",&y)!=1) /*conversie in real*/
      break;              /*s-au terminat numerele*/
    *px++=y;
    j++;
  }
```

```
  return j;
}
```

9.3. Tablouri de pointeri.

Un tablou de pointeri este definit prin: **`tip *nume[dimensiune];`**

Exemplul 27: ***Să se sorteze o listă de nume.***

Folosirea unui tablou de şiruri de caractere este lipsită de eficienţă, deoarece şirurile sunt de lungimi diferite. Vom folosi un tablou de pointeri la şiruri de caractere. Prin sortare nu se vor schimba şirurile de caractere, ci pointerii către acestea.

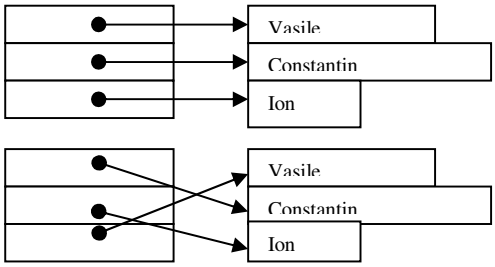

Fig.9.1. Sortarea şirurilor de caractere

Citirea şirurilor de caractere presupune:

- rezervarea de spaţiu pentru şiruri
- iniţializarea tabloului de pointeri cu adresele şirurilor

Pentru rezervarea de spaţiu se foloseşte funcţia **`char *strdup(char *s);`** care:

- salvează şirul indicat de **`s`** într-o zonă de memorie disponibilă, alocată dinamic
- întoarce un pointer către zona respectivă sau **`NULL`**.

Citirea numelor este terminată prin **`EOF`**. Funcţia de citire întoarce numărul de linii citite:

```
int citire(char *tabp[]){
   int j=0;
   char tab[80];
   while(1) {
      gets(tab);
      if(tab==NULL)
         break;
      tabp[j]=strdup(tab);
   }
   return j;
}
```

Sortarea o vom realiza cu algoritmul bulelor: dacă şirul de nume ar fi ordonat, atunci două nume consecutive s-ar afla în relaţia **<** sau **==**. Vom căuta aşadar relaţiile **>**, schimbând de fiecare dată între ei pointerii corespunzători (schimbare mai eficientă decât schimbarea şirurilor). Se fac mai multe parcurgeri ale listei de nume; la fiecare trecere, o variabilă martor – **`sortat`**, iniţializată la 1 este pusă pe 0, atunci când se interschimbă doi pointeri. Lista de nume va fi sortată în momentul în care în urma unei parcurgeri a listei se constată că nu s-a mai făcut nici o schimbare de pointeri.

```
void sortare(char *tp[], int n) {
  int j, sortat;
```

```
  char *temp;
  for(sortat=0; !sortat;){
    sortat=1;
    for(j=0;j<n-1;j++)
      if(strcmp(tp[j],tp[j+1])>0){
        temp=tp[j],
        tp[j]=tp[j+1],
        tp[j+1]=temp,
        sortat=0;
      }
  }
}
void afisare(char *tp[], int n){
  int j;
  for (j=0; j<n; j++)
    if(tp[j])
      puts(tp[j]);
}
int main()
{ int n;
  char *nume[100];
  n=citire(nume);
  sortare(nume,n);
  afisare(nume,n);
}
```

Exemplul 28: *Definiţi o funcţie, având ca parametru un întreg, reprezentând o lună, care întoarce (un pointer la) numele acelei luni*

```
char *nume_luna(int n)
{ static char *nume[]={"Luna inexistenta","Ianuarie", "Februarie",
         "Martie","Aprilie","Mai","Iunie","Iulie","August",
         "Septembrie","Octombrie", "Noiembrie","Decembrie"};
  return(n<1||n>12)?nume[0]:nume[n];
}
```

Exemplul 29: *Scrieţi un program care extrage cuvintele distincte dintr-un text, scriind în dreptul fiecărui cuvânt numărul de apariţii ale acestuia în text.*

Pentru separarea cuvintelor din text vom folosi funcţia **strtok()**. Un cuvânt separat din text este mai întâi căutat în tabloul de pointeri la cuvintele separate **cuv**, şi este inserat acolo, în caz că nu este găsit; dacă este găsit este crescut contorul de apariţii al cuvântului.

```
#include <stdio.h>
#include <string.h>
#define NC 100

/*cauta sirul p in tabloul de siruri cuv
  intoarce pozitia in care se afla p in cuv sau -1 */

int exista(char* p, int nc, char* cuv[]){
  int j=0;
```

```
  for(int j=0; j<nc; j++)
       if(strcmp(p,cuv[j])==0) return j;
  return -1;
};
int main(){
     char sep[]=" .,:;-()\t\n";    /*separatori intre cuvinte*/
     char linie[80];          /*tampon pentru citirea unei linii*/
     char *cuv[NC], ap[NC], *p;    /*tablou de cuvinte*/
     int nc=0, poz;
     freopen("text.txt", "r", stdin);
     while(gets(linie))
          for(p=strtok(linie,sep); p; p=strtok(0,sep))
               if((poz=exista(p, nc, cuv))==-1){
                    cuv[nc]=strdup(p);
                    ap[nc++]=1;
               }
               else
                    ap[poz]++;
     printf("numar cuvinte distincte %3d\n", nc);
     for(int i=0; i<nc; i++)
          printf("%10s %2d\n", cuv[i], ap[i]);
}
```

9.4. Probleme rezolvate.

1. Se citesc mai multe şiruri de caractere reprezentând numere lungi în baza 16, ce pot avea până la 100 de cifre. Să se calculeze suma acestora, ignorând numerele incorecte.

Indicaţie: Pentru a aduna două numere lungi păstrate în două şiruri de caractere, se inversează în prealabil cifrele lor, astfel încât prima cifră să fie cea mai puţin semnificativă şi se transformă apoi caracterele cifre hexadecimale în numere întregi de la 0 la 15. Se extinde apoi cu zerouri, numărul lung cel mai scurt, aducându-l la lungimea celuilalt şi se face adunarea rang cu rang, cu propagarea transportului:

$$sum = s_i + a_i + t$$
$$s_i = sum \% 16$$
$$t = sum / 16$$

Dacă în final rămâne transport, acesta devine cifra cea mai semnificativă a sumei..
Numerele din rangurile sumei se transformă în cifre hexadecimale şi se inversează aceste cifre.

1. iniţializare număr lung sumă
2. buclă citire numere lungi
   - 2.1. verificare corectitudine număr lung citit
   - 2.2. conversie caractere cifre hexa în numere
   - 2.3. inversare cifre număre lung
   - 2.4 completare număr mai scurt cu zerouri
   - 2.5. adunare pe ranguri cu propagare transport
   - 2.6. completare suma cu ultimul transport
   - 2.7. inversare cifre
   - 2.8. conversie numere hexa în caractere
3. afişare număr lung sumă

2. O fracţie raţională are numărătorul şi numitorul numere lungi în baza 10, având până la 100 cifre. Se cere să se simplifice fracţia cu divizori întregi primi, până la o limită dată **n**.
Numerele lungi se citesc ca două şiruri de caractere, din primele două linii; a treia linie conţine valoarea lui **n**.
Programul va afişa fracţia nesimplificată şi fracţia simplificată.

Indicaţie: Se va defini o funcţie care împarte un număr lung **a**, cu un întreg **d** şi calculează câtul **c** – un număr lung şi restul – un întreg mai mic ca **d**.

$$\frac{a_0 a_1 \cdots a_{n-1}}{d} = c_0 c_1 \cdots c_{n-1} + \frac{r}{d}$$

rest deîmpărţit împărţitor cât

```
int divlung(char *a, int d, char *c);
```

În prealabil, caracterele cifre ale deîmpărţitului se transformă în întregi.
Funcţia simulează împărţirea cifră cu cifră; într-un pas al împărţirii:
- se adaugă la restul parţial (iniţializat la 0) cifra curentă a deîmpărţitului: **rp=10*rp+a$_i$**
- se calculează cifra curentă a câtului: **c$_i$ = rp / d**
- se actualizează restul parţial: **rp = rp % d**

Ultimul rest parţial este restul împărţirii.
Câtul **c$_0$c$_1$…c$_{n-1}$** are aceeaşi lungime cu deîmpărţitul, dar cifrele sale cele mai semnificative pot fi 0.
În final, cifrele întregi ale câtului se transformă în caractere cifre.

1. încercare divizori primi
2. simplificare cu divizorul comun
3. afişare fracţie simplificată
4. funcţia divlung()
   - 4.1. conversie caractere în cifre
   - 4.2. simulare împărţire cifră cu cifră
   - 4.3. conversie cifre în caractere

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int divlung(char*, int, char*);
int main(){
  char a[100], b[100], ca[100], cb[100];
  int i, n;
  gets(a); puts(a);
  gets(b); puts(b);
  scanf("%d", &n);
  for(i=0; i<=n; i++)
    if(divlung(a, i, ca)==0 && divlung(b, i, cb)==0){
       strcpy(a, ca);
       strcpy(b, cb);
    }
  printf("%s\n", a);
```

```
  printf("%s\n", b);
  system("PAUSE");
  return 0;
}

int divlung(char *a, int d, char *c){
  int i, l, rp = 0;
  l = strlen(a);
  for(i=0; i < l; i++)
     a[i] -= '0';
  for(i=0; i<l; i++){
     rp = 10*rp + a[i];
     c[i] = rp / d;
     rp %= d;
  }
  for(i=0; i < l; i++)
     c[i] += '0';
  return rp;
}
```

Probleme propuse (Şiruri de caractere).

1. Scrieţi o funcţie C care stabileşte dacă un şir de caractere dat ca parametru reprezintă un palindrom. Pentru aceasta este necesar ca primul şi ultimul caracter din şirul de caractere să fie egale, al doilea şi penultimul, ş.a.m.d.
Funcţia întoarce 1 dacă şirul de caractere este un palindrom şi 0 în caz contrar.

2. Un text citit de pe mediul de intrare reprezintă un program C. Să se copieze pe mediul de ieşire, păstrând structura liniilor, dar suprimând toate comentariile.

3. Dintr-un text, citit de pe mediul de intrare, să se separe toate cuvintele, plasându-le într-un vector cu elemente şiruri de caractere de lungime 10 (cuvintele mai scurte se vor completa cu spaţii libere, iar cele mai lungi se vor trunchia la primele 10 caractere.
Se vor afişa elemenetele acestui tablou, câte 5 elemente pe o linie, separate între ele prin 2 asteriscuri.

4. Dintr-un text citit de pe mediul de intrare să se afişeze toate cuvintele care conţin cel puţin 3 vocale distincte.

5. Scrieţi un program care citeşte şi afişează un text şi determină numărul de propoziţii şi de cuvinte din text. Fiecare propoziţie se termină prin punct, iar în interiorul propoziţiei cuvintele sunt separate prin spaţii, virgulă sau liniuţă.

6. De pe mediul de intrare se citeşte un text format din cuvinte separate prin spaţii libere. Să se afişeze acest text la ieşire, păstrând structura liniilor şi scriind în dreptul liniei cel mai lung cuvânt din linie. Dacă mai multe cuvinte au aceeaşi lungime maximă, va fi afişat numai primul dintre ele.

7. Scrieţi un program care citeşte de la intrarea standard cuvinte, până la întâlnirea caracterului punct şi afişează câte un cuvânt pe o linie, urmat de despărţirea acestuia în silabe. Se utilizează următoarele reguli de despărţire în silabe:

- o consoană aflată între două vocale trece în silaba a doua
- în cazul a două sau mai multe consoane aflate între două vocale, prima rămâne în silaba întâia, iar celelalte trec în silaba următoare.
- Nu se iau în considerare excepţiile de la aceste reguli.

8. Să se transcrie la ieşire un text citit de la intrarea standard, suprimând toate cuvintele de lungime mai mare ca 10. Cuvintele pot fi separate prin punct, virgulă sau spaţii libere şi nu se pot continua de pe o linie pe alta.

9. Scrieţi un program care citeşte de la intrarea standard un text terminat prin punct şi îl transcrie la ieşirea standard, înlocuind fiecare caracter **`*'** printr-un număr corespunzător de spaţii libere care ne poziţionează la următoarea coloană multiplu de 5. Se va pastra structura de linii a textului.

10. Scrieţi un program pentru punerea în pagină a unui text citit de la intrarea standard. Se fac următoarele precizări:

- cuvintele sunt separate între ele prin cel puţin un spaţiu
- un cuvânt nu se poate continua de pe o linie pe alta
- lungimea liniei la ieşire este `N`
- lungimea maximă a unui cuvânt este mai mică decât **`N / 2`**
  In textul rezultat se cere ca la început de linie să fie un început de cuvânt, iar sfârşitul de linie să coincidă cu sfârşitul de cuvânt. În acest scop, spaţiile se distribuie uniform şi simetric între cuvinte. Face excepţie doar ultima linie. Caracterul punct apare doar la sfârşit de text.

11. Modificaţi funcţia **`strind()`** astfel încât să întoarcă în locul indexului un pointer şi **`NULL`** în caz că că subşirul **`s`** nu apare în şirul destinaţie **`d`** (adică scrieţi funcţia **`strstr()`**).