Șiruri de caractere

De la început trebuie precizat că este foarte important ca înainte de această lecție să se fi parcurs și înțeles foarte bine două materiale: cel dedicat tipului de date `char` și cel dedicat pointerilor.

Se pune problema lucrului in limbaj cu altceva decât cu numere. Un prim pas l-am văzut: folosirea tipului `char` pentru lucrul cu simboluri. Mergând mai departe, vom vedea aici cum utilizăm în limbaj cuvinte. Acestea sunt de fapt șiruri de simboluri. Soluția de a lucra cu cuvinte chiar așa a apărut: un șir de simboluri este de fapt un vector cu elemente de tip `char`.

Așa cum tipul `char` are facilități duble să spunem (tip de date întreg, ca și celelalte, dar și suport pentru a folosi simboluri) și vectorii de `char` se pot folosi, pe de o parte ca orice alt fel de vectori, dar au fost create facilități pentru a putea fi tratați ca un tot unitar și astfel să memorăm cu un astfel de vector un cuvânt, sau o propoziție.

Cum s-au realizat astfel de lucruri?

Anumite funcții de bibliotecă, precum acelea de citire/afișare, dar și multe altele speciale, permit tratarea vectorilor de tip `char` în mod special, ca pe un întreg și nu doar la nivel de element.

Aceste funcții prelucrează mai multe caractere de la începutul vectorului și anume, pe toate care încep la poziția `0` și până la prima poziție unde se află caracterul special cu codul ASCII `0`, numit și caracter de final de șir (acestui caracter îi vom mai spune și `NULL`, fără să îl confundăm ca semnificație cu constanta pointer vid pe care am numit-o la fel).

Ce semnifică această convenție?

Noi știm că folosirea unui vector implică de regulă rezervarea de memorie pentru un maxim necesar, la declarare, apoi însoțirea vectorului de o dimensiune logică, un `n`, ce indică până unde folosim elemente la testul curent. În cazul funcțiilor care prelucrează automat șiruri de caractere, convenția să se meagă până la caracterul `NULL` este modul prin care evităm folosirea unei variabile din afară pentru a da consistență informațiilor din tablou. În plus, elementele utile din vector încep cu acela de pe poziția `0`, chiar dacă suntem poate obișnuiți de la tablouri să lucrăm începând cu indicele `1`.

Preluarea de la tastatură și afișarea pe ecran a șirurilor de caractere.

Fie codul:

```
int v[100];
char s[100];
…
cin>>v;
cin>>s;
```

Ne este cunoscut faptul că o linie de cod de forma `cin>>v`, cu `v` tablou de `int`-uri, produce eroare de compilare. Noi nu putem cere preluarea de la tastatură a unui vector de `int`-uri așa tot deodată ci element cu element.

În schimb, convenția face ca `cin>>s` să meargă. Care este efectul?

Dacă, de exemplu, se introduce de la tastatură secvența: `aB0*c` și se apasă enter, se întâmplă:

`s[0] = 'a';` (adică `97`)
`s[1] = 'B'` (adică `66`)

`s[2] = '0'` (adică `48`)
`s[3] = '*'` (adică `42`)
`s[4] = 'c'` (adică `99`)
`s[5] = NULL` (adică `0`)

Așadar, prin simpla scriere `cin>>s` se respectă convenția.

Citirea cu `cin` exact ca mai sus nu permite preluarea caracterelor albe (cum ar fi spațiul) iar pentru a include și astfel de caractere în șiruri la citire vom folosi `cin.get` și vom vedea cum se face asta ulterior.

Pentru afișare se folosește `cout<<vector;`

Evident că pentru declarațiile de mai sus, `cout<<v` produce eroare de compilare pe când `cout<<s` este în regulă și are următorul efect: se afișează unul lângă altul, simbolurile caracterelor, începând cu poziția `0` și până înaintea poziției pe care se află `NULL`. Deci, pentru exemplul de mai jos s-ar afișa ce s-a introdus, adică `aB0*c`.

Putem oare pune în cod o linie de forma următoare?

```
cout<<s+2;
```

Răspunsul este DA și, chiar dacă lucrurile par puțin complicate, acum este momentul de a le înțelege mai bine.

`cout` primește de fapt un pointer la `char` (asta este de fapt numele unui vector) și tipărește simbolurile tuturor carecterelor care se găsesc începând cu adresa memorată în pointer și până la `NULL`. `s+2` este tot un pointer pentru că noi știm că este permis operatorul `+` între un pointer și un număr, rezultatul fiind un pointer.

Așadar efecul instrucțiunii de afișare de mai sus este de a scrie pe ecran simbolurile caracterelor aflate începând cu două componente dincolo de cea de la adresa din `s` și până la primul `NULL`. Adică `0*c`.

Să analizăm următorul exemplu:

`char s[100];`	
`cin>>s;`	Se introduce `abcdef` și enter
`cout<<s`	Se afișează: `abcdef`
`s[2] = '*';`	
`cout<<s;`	Se afișează: `ab*def`
`s[4] = 0;`	
`cout<<s;`	Se afișează: `ab*d`. Adresa de unde să înceapă afișarea noi nu am schimbat-o însă am pus `NULL` mai în față și convenția este că se tipăresc simboluri până la primul `NULL`.
`s[4] = 'x';`	
`cout<<s;`	Se afișează: `ab*dxf`. De aici trebuie să tragem o concluzie foarte importantă: dacă anterior am pus `NULL` pe poziția `4`, acest lucru nu a afectat și alte poziții din șir. Deci pe poziția `5` a rămas `f` și pe poziția `6` `NULL` initial etc. Acum revenind și ștergând practic acel `NULL` pus de noi (l-am înlocuit cu alt caracter) afișarea se va opri la `NULL` de pe poziția a `6`-a.

O concluzie este că pentru a citi de la tastatură un șir de caractere care nu conține spații folosim `cin` cu parametru vectorul de caractere iar pentru a-l afisa pe ecran folosim `cout`, de asemenea cu parametru vectorul șir de caractere. De remarcat că la afișare, dacă șirul conține spații acestea se vor afișa.

Exemplu

```
char s[100];
cin>>s; /// introduc abcdef
s[2] = ' ';
cout<<s; /// se afișează ab def
```

Discuția mai detalită despre modul de citire a unui șir care poate conține și spații o facem la secțiunea dedicată citirii mai multor șiruri aflate pe rânduri diferite. Acum prezentăm modul în care facem asta dacă dorim citirea unui singur astfel de șir:

```
cin.getline(sir, dimensiune);
```

Apelul de mai sus face să se aștepte de la tastatură caracterele șirului (inclusiv spații) și memorarea acestora în sir (care este un pointer la `char`, adică un vector de caractere). Sunt luate în calcul maxim "dimensiune" caractere.

Exemplu:

```
char s[101];
cin.getline(s, 101); /// se introduce: ziua aceasta frumoasa
cout<<s; /// se afișează: ziua aceasta frumoasa
cuvinte = 0;
if (s[0] != ' ')
    cuvinte++;
for (i=1;s[i]!=0;i++)
    if(s[i]!=' ' && s[i-1] == ' ')
        cuvinte++;
cout<<cuvinte;
```

Presupunând că s-au introdus doar litere mici ale alfabetului și spații și că o secvență de litere delimitată de spații sau capetele șirului formează un cuvânt, s-a calculat și afișat și numărul de cuvinte. La apelarea lui `cin.getline`, al doilea parametru se trece de regulă suficient de mare pentru a nu limita el numărul de caractere preluate.

În continuare prezentăm o problemă prin care arătăm modul de realizare a unor operații clasice pe un șir de caractere pe care îl parcurgem element cu element.

Se citește un șir de maxim `100` de caractere despre care se știe că poate conține doar litere ale alfabetului (mari sau mici) cifre și semne dintre: `+`, `-`, `*`, `(`, `)`, `[`, `]`.

Să se afișeze numărul de:

- Litere mici
- Litere mari sau mici
- Cifre
- Vocale litere mici

De la capitolul dedicat tipului `char` cunoaștem că testul ca un caracter să fie literă mică este unul de apartenență a unui număr la un interval, testul de literă oarecare este unul de apartenență la unul dintre două intervale, testul de cifră este, de asemenea, unul de apartenență la un interval iar la testul de vocală literă mică va trebui să verificăm egalitate cu fiecare dintre cele cinci vocale întrucât acestea nu formează o secvență

de coduri consecutive. Vom parcurge așadar caracter cu caracter șirul și vom face testul pentru fiecare caracter în parte.

```
#include <iostream>
using namespace std;
char s[101];
int litereMici, litere, cifre, vocale;
int main() {
    cin>>s;
    for (int i=0;s[i]!=0;i++) {
        if (s[i] >= 'a' && s[i] <= 'z')
            litereMici++;
        if ((s[i]>='a' && s[i]<='z') || (s[i]>='A' && s[i]<='Z'))
            litere++;
        if (s[i] >= '0' && s[i] <= '9')
            cifre++;
        if (s[i]=='a'||s[i]=='e'||s[i]=='i'||s[i]=='o'||s[i]=='u')
            vocale++;
    }
    cout<<litereMici<<" "<<litere<<" "<<cifre<<" "<<vocale;
}
```

Remarcați condiția de terminare a repetiției, sau, altfel spus, de terminare a șirului: `s[i] != 0`. Veți mai întâlni și `s[i]!=NULL` sau `s[i]<strlen(s)`, așa cum se poate vedea din cele ce urmează.

Iată și o altă problemă rezolvată utilă pentru buna înțelegere a modului de memorare a unui șir de caractere dar și pentru reactualizarea unor noțiuni legate de pointeri.

Se citește un cuvânt, alcătuit din litere mici ale alfabetului (maxim `20`) și se cere afișarea, câte unul pe rând, mai întâi a sufixelor cuvântului dat și apoi a prefixelor.

Exemplu

Date de intrare	Date de ieșire
pion	pion ion on n pion pio pi p

Rezolvare

```
#include <iostream>
using namespace std;
char s[21];
int i;
int main () {
    cin>>s;
    for (i=0;s[i]!=0;i++)
        cout<<s+i<<"\n";
    for (i--;i>=0;i--) {
        cout<<s<<"\n";
        s[i] = NULL;
```

```
    }
}
```

Să analizăm atent și cu răbdare codul de mai sus.

Primul `for` afișează sufixele, trimițând lui `cout` ca pointer de început al afișării toate adresele de după cea memorată în `s`. Deci, inițial avem `cout<<s+0` (adică se tipărește tot șirul), apoi `cout<<s+1` (se tipărește fără primul caracter) etc.

Al doilea for tipărește numai din adresa de început însă pune `NULL` în locul caracterelor șirului începând de la dreapta spre stânga. Inițial se tipărește tot șirul, apoi se înlocuiește ultimul caracter util cu `NULL`, se tipărește iarăși (acum `NULL` este mai aproape de început), iarăși se pune `NULL` în locul ultimului caracter util, și tot așa.

Constante șir de caractere

Foarte des am folosit până acum, la afișări, noțiunea de mesaj. De exemplu, chiar la prezentarea lui cout am spus că o putem folosi cu argumente care pot fi expresii sau mesaje (simboluri plasate între ghilimele). Este momentul să spunem că un șir de simboluri puse între ghilimele reprezintă o constantă șir de caractere.

Exemplu

`cout<<"Azi este soare";`	Aici `cout` are ca parametru o constantă șir de caracrtere.

`cout<<"Azi este\nsoare";`	Se tipărește:
	`Azi este`
	`soare`
	Observăm că într-o constantă șir de caractere putem avea simboluri dar și secvențe escape.

Nu trebuie să confundăm constantele șir de caractere cu constantele de tip `char`. Facem următoarea analiză între `'a'` și `"a"`. Asemănarea ar fi că ambele produc tipărirea pe ecran a simbolului `a` dacă sunt transmise lui `cout`. Însă sunt diferențe. De exemplu `'a'` reprezintă un număr, codul `ASCII` al lui `a`. Putem folosi constanta în orice calcule aritmetice. Pe de altă parte, `"a"` este mai degrabă un vector format din două numere, `97` (codul lui `a`) și `NULL`, adică simbolul de final de șir. Deci pe ea nu o putem folosi, de exemplu, în calcule matematice.

Funcții predefinite pentru lucru cu șiruri de caractere din biblioteca `<cstring>`

Am arătat că vectorii de caractere sunt interpretați în particular dacă sunt transmiși spre rutine de intrare/ieșire. Majoritatea funcțiilor care îi tratează special, și cu ajutorul cărora facem diverse prelucrări la nivel de cuvânt, se află în biblioteca `cstring`. Sunt foarte multe funcții acolo și este greu de ținut minte pe toate. Noi vom prezenta câteva, foarte des întâlnite și a căror înțelegere ne ajută să căutăm în bibliotecă și altele ce ne-ar putea fi utile într-o anume situație.

Funcțiile le vom prezenta folosind convenția:

```
tip_rezultat nume_funcție (lista de parametri, separați prin virgulă)
```

1. **Funcția de determinare a lungimii șirului.**

```
int strlen(char s[])
```

Notația dintre paranteze indică prezența unui singur parametru, șir de caractere. Rezultatul este întreg și reprezintă lungimea șirului, adică prima poziție pe care se găsește caracterul `NULL` (cel care are codul ASCII `0`) la o căutare începând cu poziția `0`.

Noi am prezentat parcurgerea caracter cu caracter a unui șir astfel:

```
for (int i=0;s[i]!=0;i++)
  …
```

Acum ne putem da seama că secvența de mai sus este echivalentă cu:

```
for (int i=0;i<strlen(s);i++)
  …
```

Observați că avem operatorul < și nu <=. Adică avem caractere efectiv între pozițiile `0` și `strlen(s)-1` inclusiv.

O altă remarcă foarte importantă este că un apel al funcției `strlen` nu dă rezultatul instant ci îl obține prin parcurgerea caracter cu caracter a șirului, în scopul găsirii lui `NULL`. Așadar e ca și cum secvența de mai sus ar avea for în `for`. Recomandarea este fie să folosim modul de parcurgere prezentat inițial fie să apelăm `strlen` la început, să stocăm rezultatul într-o variabilă și să o folosim pe aceata în `for`, ca în continuare:

```
int len = strlen(s);
for (int i=0;i<len;i++)
  …
```

Funcția se poate aplica atât pentru șiruri memorate în vectori dar și pentru șiruri constante. De exemplu, un apel de forma `cout<<strlen("info");` are ca efect afișarea pe ecran a valorii `4`.

2. Funcția de copiere a conținuturilor

Este deseori necesar ca unei variabile, de orice tip, să îi facem o copie în altă variabilă. Să vedem ce ar însemna asta în cazul șirurilor de caractere.

Fie codul:

```
char s[100];
char t[100];
cin>>s; /// introducem abcdef
```

Dorim ca în `t` să ajungă același conținut ca în `s`. În general apare impulsul de a folosi atribuirea, adică să scriem `t=s`. Dar nu obținem efectul dorit. Aici `s` și `t` sunt pointeri și ceea ce am cere noi este de a copia adresa care se află în `s` în variabila `t`. Nici măcar acest lucru nu se poate întâmpla întrucât pentru ponterii declarați ca și `s` nu este permisă modificarea valorii (ei sunt legați pe toată durata executării programului de zona de memorie care se alocă și pe a cărei adresă trebuie să o păstreze).

Pentru copierea conținuturilor avem o funcție de bibliotecă:

```
strcpy(char sirDestinatie[], char sirSursa[])
```

Efectul este că se vor copia caracter cu caracter cele care încep la poziția `0` în `sirSursa`, în `sirDestinatie` începând tot cu poziția `0`. Se copiază inclusiv `NULL`. Este ceva echivalent cu:

```
for (i=0;i<=strlen(sirSursa);i++)
    sirDestinatie[i]=sirSursa[i];
```

Dar noi facem asta scriind doar apelul funcției.

Exemple (considerăm că avem `s` și `t` declarați ca mai sus ca vectori de caractere)

`strcpy(s, "floare");` `cout<<s;`	Se tipărește pe ecran floare. Aici al doilea parametru este o constantă șir de caractere si obtinem efect de forma: `s[0] = 'f'; s[1] = 'l';... s[6]=0;`
`strcpy(s, "floare");` `strcpy(t, "utur");`	Se afișează flutur. Conținutul de la adresa `t` se copiază la nivel de caracter începând de la adresa `s+2`

`strcpy(s+2, t);`	

`cin>>s>>t;`
`char aux[101];`
`strcpy(aux, s);`
`strcpy(s, t);`
`strcpy(t, aux);`
	Aici arărăm cum interschimbăm conținuturile celor două șiruri. Folosim un șir auxiliar. Observați că la `cin` avem parametri două șiruri de caractere. La introducerea datelor cele două secvențe de simboluri trebuie sepate prin spații sau/și enter. Este momentul sa întărim faptul că la citirea cu `cin` simplu caracterele albe (`spațiu`, `enter`, `tab`) se consideră separatori. Vă puteți da seama de asta dacă vă gândiți acum că așa se întâmplă când citim numere.

La prezentarea funcției `strcpy` nu am indicat tipul rezultatului. Am făcut intenționat asta pentru a ne concentra înțelegerea pe faptul că se copiază conținuturile. Funcția returnează pointerul memorat în șirul destinație.

Deci putem scrie ceva de genul: `cout<<strcpy(s, "abcd");` Efectul este că se memorează `abcd` în șirul `s` dar și că se afișează pe ecran `abcd`. De cele mai multe ori funcția o folosim însă ca instrucțiune independentă și mai rar într-o expresie în care se utilizează mai departe pointerul returnat.

3. Funcția de alipire a conținuturilor

Ca sintaxă este la fel ca funcția de copiere iar efectul este că se alipește conținutul șirului sursă la cel al șirului destinație. Operația de alipire a conținuturilor se mai numește și concatenare.

`strcat(char sirDestinatie[], char sirSursa)`

Valoarea returnată este pointer la șirul destinație, ca și la funcția `strcpy`.

Exemplu

`char s[101], t[101];`
`strcpy(s, "de");`
`strcat(s, "lemn");`
`cout<<s;`
`strcpy(t, "unt");`
`strcat(t, s);`
`cout<<t;`
	Prima dată se afișează `delemn` întrucât acesta este conținutul lui `s` obținut în urma operației de concatenare între valoarea sa anterioara `de` și șirul `lemn`.
	A doua oară se obține în `t` și apoi se afișează `untdelemn`.

`char s[101], t[101];`
`strcpy(s, "ante");`
`strcpy(t, "premergator");`
`strcat(s, t+3);`
`cout<<s;`
	Se afișează `antemergator`.
	La apelul strcat pointerul ce indică spre șirul destinație l-am obținut folosind operatorul de adunare de pointer cu număr la șirul `t`, lipind deci la s un sufix din `t`.

Deci, mai detaliat, avem: primul caracter al șirului sursă se copiază în loc de `NULL` de la finalul celui destinație iar următoarele caractere unul după altul în continuare. La final se pune `NULL`.

4. Funcția de comparare lexicografică (alfabetică) a conținuturilor.

Reamintim ce înseamnă să comparăm lexicografic două șiruri de valori: cea mai mică poziție unde valorile din cele două șiruri diferă dă rezultatul comparării, restul valorilor, aflate pe poziții mai mari cu aceasta, nu mai contează.

Exemple

`ioan` `ioan`	Șirurile sunt identice

`Ioan` `ioan`	Aici șirurile diferă chiar la prima poziție (se compară în fapt codurile literelor și `I` (mare) are cod diferit decât `i`(mic) și mai mic). Deci primul șir este mai mic lexicografic decât al doilea.
`masca` `marama`	Pe primele două poziții șirurile sunt identice iar la poziția a treia se face diferența. Codul lui `s` este mai mare decât cel al lui `r` deci primul șir este mai mare lexicografic decât aldoiea (deci nu mai contează ce este după și nici lungimile șirurilor)
`mara` `marama`	Aici șirurile sunt identice pe lungimea celui mai scurt. În acest caz cel mai scurt este considerat mai mic lexicografic. Gândiți-vă că ne putem imagina că poziția unde este `NULL` în șirul cel mai scurt corespunde unei poziții în care celălalt șir are ceva nenul și de aici rezultatul.

O comparare cu == între `s` și `t` nu e altceva decât o comparare a valorilor celor doi pointeri (așa cum știm de la capitolul dedicat pointerilor). Evident că `s` și `t` memorează adrese diferite căci primesc zone diferite de memorie ca efect al declarării. Deci chiar dacă în cele două zone conținutul ar fi identic, `s==t` ar da rezultat fals.

Așadar este necesară o funcție pentru a realiza compararea după conținut. Iat-o:

```
int strcmp(char sir1[], char sir2[]);
```

Dacă sunt identice conținuturile celor două șiruri rezultatul funcției este `0`. Dacă `sir1` are conținutul mare lexicografic decât `sir2` atunci rezultatul este pozitiv, în celălalt caz rezultatul fiind negativ. Se recomandă ca rezultatul funcției să fie așadar comparat cu `0`.

Iată în continuare un alt set de probleme rezolvate

1. Se citesc de la tatatură două siruri formate din litere mici ale alfabetului despre care se cunoaște că sunt de lungimi diferite. Să se afișeze pe un rând al ecranului, separate prin spațiu cele două șiruri în ordine crescătoare a lungimii și pe alt rând, separate prin spațiu, cele două șiruri în ordine alfabetică.

Exemplu

Date de intrare	Date de ieșire
`masca marama`	`masca marama` `marama masca`

Rezolvare

Pentru afișarea după primul criteriu se folosește funcția strlen la comparare iar în al doilea caz se folosește `strcmp`.

```
#include <iostream>
#include <cstring>
using namespace std;
char s[21], t[21];
int main () {
    cin>>s>>t;
    if (strlen(s) < strlen(t))
        cout<<s<<" "<<t<<"\n";
    else
        cout<<t<<" "<<s<<"\n";

    if (strcmp(s, t) < 0)
        cout<<s<<" "<<t<<"\n";
    else
        cout<<t<<" "<<s<<"\n";
```

```
}
```

2. Se citește de la tastatură un cuvânt care conține litere mici ale alfabetului. Să se șteargă litera aflată pe o poziție dată `p`.

Exemplu

Date de intrare	Date de ieșire
avere 1	aere

Rezolvare

Versiunile mai vechi ale limbajului permiteau realizarea operației printr-o instrucțiune de copiere de conținuturi de forma: `strcpy(s+p,s+p+1);` Versiunile recente recomandă evitarea suprapunerii zonelor de copiat în cazul folosirii funcției `strcpy`. Vom utiliza o zonă auxiliară de memorie în care copiem sufixul ce va rămâne (adică de la poziția `p+1` până la final) apoi aducem de acolo în `s` începând cu poziția `p`.

```
#include <iostream>
#include <cstring>
using namespace std;
char s[21];
char aux[21];
int p;
int main () {
    cin>>s;
    cin>>p;
    if (p>=0 && p < strlen(s)) {
        strcpy(aux, s+p+1);
        strcpy(s+p, aux);
    }
    cout<<s;
}
```

3. Să se insereze într-un șir `s`, la poziția `p`, caracterul `c`.

Exemplu

Date de intrare	Date de ieșire
aere 1 v	avere

Rezolvare

Procedăm asemănător problemei anterioare, urmând pașii:

- Mutăm sufixul în altă parte `strcpy(aux, s+p);`
- Memorăm caracterul nou pe poziția dată `s[p] = c;`
- Aducem sufixul după caracterul inserat `strcpy(s+p+1, aux);`

```
#include <iostream>
#include <cstring>
using namespace std;
char s[21];
char aux[21];
```

```
int p;
char c;
int main () {
    cin>>s;
    cin>>p;
    cin>>c;
    if (p>=0 && p <= strlen(s)) {
        strcpy(aux, s+p);
        s[p] = c;
        strcpy(s+p+1, aux);
    }
    cout<<s;
}
```

4. Se citește de la tastatură un șir de caractere despre care știm că este format doar din simbolurile cifră `7` și `9`. Să se înlocuiască fiecare simbol cu cifra romană corespunzătoare, scrisă cu litere mari.

Exemplu

Date de intrare	Date de ieșire
`77979`	`VIIVIIIXVIIIX`

Rezolvare

Problema admite și alte soluții decât cea prezentată, noi am dorit să atragem atenția asupra următoarei abordări:

- Folosim un șir auxiliar pe care îl inițializăm cu șirul vid. Putem face asta cum se vede în cod: cu `strcpy`, copiind "" (fără spații) sau memorând valoarea `0` (`NULL`) chiar pe poziția `0`.
- Parcurgem șirul dat și în funcție de simbolul întâlnit concatenăm la șirul auxiliar secvența corespunzătoare.
- Copiem în șirul initial rezultatul obținut în șirul auxiliar.

```
#include <iostream>
#include <cstring>
using namespace std;
char s[21], t[101];
int main () {
    cin>>s;
    strcpy(t, ""); /// sau t[0] = 0;
    for (int i=0;s[i]!=0;i++)
        if (s[i] == '7')
            strcat(t, "VII");
        else
            strcat(t, "IX");
    strcpy(s, t);
    cout<<t;
}
```

5. Rezolvăm aceeași problemă ca mai sus dar în acest caz șirul poate să mai conțină și litere mici ale alfabetului pe lângă simbolurile `7` și `9`. Dorim deci să înlocuim pe `7` și `9` cu cifrele romane corespunzătoare dar literele mici să nu se schimbe.

Rezolvare

Ne-am putea gândi imediat ca în cazul în care `s[i]` este cifră să executăm `strcat(t, s[i])`. Acest lucru este însă incorect sintactic deoarece al doilea parametru de la `strcat` este de tip întreg (char simplu) și nouă ne trebuie un pointer (adică un vector de `char`-uri). Putem să ne adaptăm în felul următor: folosim un șir de caractere pe care îl pregătim să stocheze un singur caracter util:

- Memorăm de la început pe poziția `1` valoarea de final de șir, adică `0`.
- Când `s[i]` este literă copiem pe `s[i]` pe poziția `0` în șirul auxiliar.
- Putem acum să folosim șirul auxiliar ca al doilea parametru la `strcat`.

```
char aux[2];
aux[1] = 0;
for (int i=0;s[i]!=0;i++)
    if (s[i] == '7')
        strcat(t, "VII");
    else
        if (s[i] == '9'
            strcat(t, "IX");
        else {
            aux[0] = s[i];
            strcat(t, aux);
        }
```

Funcții de căutare

Sunt două cele pe care ne vom concentra, asemănătoare ca sintaxă și principiu de funcționare, una caută un caracter întru-un șir de caractere iar cealaltă un șir de caractere ca secvență în altul. Ambele returnează un pointer, deci nu o poziție, semnificând adresa primei apariții.

`strchr(char sir[], char caracter)`

Această funcție returnează adresa primului loc unde caracterul apare în șir. În cazul în care caracterul nu apare în șir rezultatul este `NULL`.

`strstr(char sir1[], char sir2[])`

Diferența este că al doilea parametru este tot pointer ca și primul, spre deosebire de `strchr` unde al doilea parametru este un `char`, adică un număr. Funcția returnează adresa primului loc unde `sir2` începe ca secvență în `sir1` sau `NULL` dacă `sir2` nu apare ca secvență în `sir1`.

Exemple

Considerăm în toate exemplele că avem:

```
char s[101], t[101];
char c;
int a;
char *p;
```

`strcpy(s, ”farmec”);` `p = strchr(s, ’a’);`	Variabila pointer `p` va primi o valoare diferită de `NULL` deoarece caracterul `a` apare în șirul `s`.

`strcpy(s, ”farmec”);` `p = strchr(s, 97);`	Acest exemplu este identic cu anteriorul, diferă doar modul de specificare a caracterului ca parametru la funcție (aici am ales să scriem codul ASCII al simbolului a ca o constantă numerică).

`strcpy(s, ”farmec”);`
`p = strchr(s, ’a’);`
`cout<<p;`
	Se afișează pe ecran armec. Pas cu pas, justificarea ar fi:
	- Se returnează un pointer, adresa la care apare `‘a’` în `s`.
	- Se copiază adresa în variabila pointer `p`.
	- Afișarea cu argument un pointer la `char` am văzut că produce tipărirea tuturor caracterelor începând cu adresa din pointer până prima dată unde se întâlnește `NULL`.

`strcpy(s, ”farmec”);`
`p = strchr(s, ’a’);`
`cout<<p-s;`
	Aici se tipărește `1`, adică poziția primei apariții a caracterului în șir. Ne-am folosit de operația de diferență a pointerilor care știm că este permisă și care oferă numărul de componente dintre cei doi pointeri, adică exact ce ne interesează pe noi. Diferența a doi pointeri este deci inversa operației de adunare a unui pointer cu un număr.

`strcpy(s, ”destabilizata”);`
`p = strchr(s, ‘a’);`
`cout<<strchr(p+1, ‘a’)-s;`
	Aici se afișează poziția celei de-a doua apariții a literei `a` în șirul dat. Dacă am fi apelat de două ori `strchr(s, ‘a’)` am fi obținut în ambele cazuri același lucru (adresa primei apariții). Noi, pentru a găsi a doua apariție, a trebuit ca al doilea apel să îl facem de la adresa ce urmează pe cea a primei apariții.

`strcpy(s, ”Antananarivo”);` `cout<<strstr(s,”na”);`	Se afișează pe ecran `nanarivo` (funcția `strstr` returnează adresa primei apariții a lui `na` în șir).
`strcpy(s,”pion”);` `cout<<strstr(s, ’o’);`	Apare eroare de compilare. În de `’o’` trebuie pus `”o”`.
`strcpy(s, ”pion”);` `p = strstr(s, ”in”);`	`p` va avea valoarea `NULL`. Chiar dacă `i` și `n` se găsesc ambele în `s` și chiar în aceeași ordine, căutarea se făcea cu succes doar dacă apăreau în aceeași ordine pe poziții consecutive.

Probleme rezolvate

Acestea trebuie privite ca aplicații teoretice de mare importanță în rezolvarea altor task-uri.

1. Se citește un șir format din litere mici ale alfabetului. Să se determine numărul de vocale.

Rezolvare

Am prezentat în capitolul despre tipul char modul de testare dacă un caracter este vocală. Făceam testul comparând pe rând cu fiecare dintre cele `5` vocale. Acum arătăm cum facem testul mai elegant folosind funcția strchr.

- Construim înainte un șir de caractere cu cele `5` vocale
- Caracterul de testat este căutat în acest șir

Pentru a forma șirul cu vocalele putem să declarăm un vector de caractere pe care îl putem inițializa chiar la declarare în modul: `char voc[] = “aeiou”;` și apoi apelăm `strchr(voc, caracter de testat)` sau direct, trimitem lui `strchr` șirul de vocale ca șir constant, `strchr(“aeiou”, caracterul de testat)`.

```
#include <iostream>
#include <cstring>
using namespace std;
char voc[] = "aeiou";
```

```
char s[101];
int i, vocale = 0;
int main() {
    cin>>s;
    for (i=0;s[i]!=0;i++)
        if (strchr(voc, s[i]) != NULL)
            vocale++;
    cout<<vocale;
    return 0;
}
```

2. Se citește un șir format din litere mari și mici ale alfabetului. Să se sransforme fiecare fiecare vocală literă mică în litera mare corespunzătoare.

Rezolvare

Știm că înre litera mică și litera mare corespunzătoare diferența este constantă indiferent despre ce literă este vorba (atât literele mici cât și cele mari au codurile cu valori consecutive). Această diferență o obținem prin expresii de forma `'a'-'A'`.

```
#include <iostream>
#include <cstring>
using namespace std;
char voc[] = "aeiou";
char s[101];
int i;
int main() {
    cin>>s;
    for (i=0;s[i]!=0;i++)
        if (strchr(voc, s[i]) != NULL)
            s[i] -= 'a'-'A';
    cout<<s;
    return 0;
}
```

3. Să se determine toate pozițiile pe care o anume literă mică apare într-un cuvânt (șir de caractere format din litere mici).

Vom prezenta pentru această problemă două soluții: una simpla și recomandată, cealaltă mai mult didactică și care ne ajută să rezolvăm ulterior task-uri mai complexe.

Soluția 1

Parcurgem șirul și facem testul pentru fiecare caracter folosind operatorul de comparare.

```
char s[101],  c;
cin>>s>>c;
for (i=0;s[i]!=0;i++)
    if (s[i] == c)
        cout<<i<<" ";
```

Soluția 2

Vom folosi funcția de căutare pe care o apelăm mereu cu adresa de început ca fiind cea de după apariția anterioară.

```
#include <iostream>
#include <cstring>
using namespace std;
char s[101], *p, *q, c;
int main() {
    cin>>s>>c;
    p = s;
    while ((q=strchr(p,c)) != NULL) {
        cout<<q-s<<" ";
        p = q+1;
    }
    return 0;
}
```

Să analizăm atent ce se întâmplă mai sus.

- `p` este adresa de la care fac următoarea căutare, inițial `p=s`
- la condiția de la `while` se întâmplă mai multe lucruri: se caută caracterul începând cu adresa din `p`, se copiază adresa găsită în `q` și se și compară aceasta cu `NULL`
- astfel, putem și să testăm dacă am găsit o nouă apariție și totodată să știm cum îl setăm pe `p` pentru următoarea căutare.

Dacă nu am fi salvat în `q` ar fi trebuit să apelăm din nou funcția de căutare între instrucțiunile lui `while` și asta ar fi însemnat calcul inutil.

4. Generalizăm problema anterioară și în loc să aflăm pozițiile pe care un caracter se află într-un șir, vom afla pozițiile la care un alt șir apare ca secvență în șirul dat.

Exemplu

Date de intrare	Date de ieșire
`Abcxyzabxyzsxyzx xyz`	`3 8 12`

Rezolvarea constă în adaptarea simplu a soluției a doua de la problema anterioară. Adică doar am declarat pe `c` ca fiind șir de caractere în loc de `char` simplu și am folosit `strstr` în loc de `strchr`.

```
#include <iostream>
#include <cstring>
using namespace std;
char s[101], *p, *q, c[101];
int main() {
    cin>>s>>c;
    p = s;
    while ((q=strstr(p,c)) != NULL) {
        cout<<q-s<<"\n";
        p = q+1;
    }
    return 0;
}
```

De remarcat că aparițiile celui de-al doilea șir se pot suprapune în primul și noi le identificăm pe toate (căutarea următoare începe doar cu un caracter mai la dreapta decât ultima găsită, nu după încheierea ultimei găsite).

5. Date fiind două șiruri de caractere, să se șteargă din primul șir toate aparițiile celui de-al doilea. Presupunem că aparițiile celui de-al doilea șir nu se suprapun în primul.

Rezolvare

Odată ce am identificat o apariție ștergerea caracterelor ei o facem în mod asemănător cu ștergerea unui singur caracter: copiem într-un șir auxiliar sufixul ce trebuie să rămână și apoi din șirul auxiliar aducem la poziția la care începe secvența de șters. Aici observăm că vom face căutarea chiar din poziția în care am găsit apariția precedentă întrucât aparițiile pot fi una după alta imediat. Așadar nu a mai fost nevoie și de pointerul auxiliar `q`.

```
#include <iostream>
#include <cstring>
using namespace std;
char s[101], *p, c[101], aux[101];
int main() {
    cin>>s>>c;
    p = s;
    while ((p=strstr(p,c)) != NULL) {
        strcpy(aux, p+strlen(c));
        strcpy(p, aux);
    }
    cout<<s;
    return 0;
}
```

6. Se citesc două șiruri de caractere formate din litere mici ale alfabetului. Să se verifice dacă ele sunt anagrame. Asta înseamnă să fie formate din exact aceleași litere, eventual dispuse în altă ordine (și o anume literă să apară de același număr de ori în ambele șiruri). Programul va afișa `0` sau `1` cu semnificația de valoare de adevăr. Pbinfo.ro, `#97`.

Exemplu

Date de intrare	Date de ieșire
`rutina unitar`	`1`

Rezolvare

Pentru această problemă vom prezenta trei soluții. Vă rog să le urmăriți atent pe toate, din fiecare fiind, evident, câte ceva de învățat.

Soluția 1.

Fiind vorba despre vectori de caractere și tipul char fiind un tip întreg, ne dăm seama că putem ordona crescător ambele șiruri și apoi testul devine să verificăm dacă sunt identice noile conținuturi.

```
#include <iostream>
#include <cstring>
using namespace std;
char s[100];
char t[100];
void sorteaza(char *s) {
    int i, j;
```

```
    char aux;
    for (i=0;i<strlen(s);i++)
        for (j=i+1;j<strlen(s);j++)
            if (s[i] > s[j]) {
                aux = s[i];
                s[i] = s[j];
                s[j] = aux;
            }
}
int main () {
    cin>>s>>t;
    if (strlen(s) != strlen(t)) {
        cout<<0;
        return 0;
    }
    sorteaza(s);
    sorteaza(t);
    if (strcmp(s, t) == 0)
        cout<<1;
    else
        cout<<0;
    return 0;
}
```

Soluția 2

Caracterele fiind de fapt numere, putem folosi vectori de frecvență cu indici caractere. Astfel, pentru fiecare șir vom folosi câte un astfel de vector determinând numărul de apariții pentru fiecare caracter. Testul nostru devine apoi echivalent cu a testa dacă sunt identici cei doi vectori de frecvență.

```
#include <iostream>
#include <cstring>
using namespace std;
char s[100];
char t[100];
int i;
int fs[130];
int ft[130];
void marcheaza(char *s, int *v) {
    for (i=0;s[i]!=0;i++)
        v[s[i]]++;
}
int main () {
    cin>>s>>t;
    if (strlen(s) != strlen(t)) {
        cout<<0;
        return 0;
    }

    marcheaza(s, fs);
    marcheaza(t, ft);
    for (i='a';i<='z';i++)
        if (fs[i] != ft[i]) {
            cout<<0;
            return 0;
        }
    cout << 1;
```

```
    return 0;
}
```

Observăm că puteam folosi un singur vector de frecvență în care să incrementăm frecvențe la primul șir și să decrementăm la al doilea. La final ar trebui să verificăm dacă vectorul nostru are toate elementele nule.

Soluția 3

Vom parcurge primul șir și caracterul curent îl căutăm în al doilea șir. Dacă nu este găsit programul va afișa `0`. Dacă îl găsim va trebui să îi ștergem apariția din al doilea șir pentru a permite ca o altă apariție a aceleiași valori în primul șir să fie asociată cu una nouă în al doilea (noi cunoaștem că funcția `strchr` caută doar prima apariție a unui caracter într-un șir). Pentru a trata și cazul ca al doilea șir să nu aibă caractere în plus, fie ne asigurăm de la început ca ambele șiruri să aibă aceeași lungime fie verificăm dacă la final al doilea șir devine vid.

```
#include <iostream>
#include <cstring>
using namespace std;
char s[100];
char t[100];
char saux[100];
int i;
char *p;
int main () {
    cin>>s>>t;
    if (strlen(s) != strlen(t)) {
        cout<<0;
        return 0;
    }
    for (i=0;s[i]!=0;i++) {
        p = strchr(t, s[i]);
        if (p == NULL) {
            cout<<0;
            return 0;
        }
        /// sterg caracterul de la adresa p din sirul t
        strcpy(saux, p+1);
        strcpy(p, saux);
    }
    if (strlen(t) == 0)
        cout<<1;
    else
        cout<<0;
    return 0;
}
```

Funcțiile strncpy, strncat și strncmp

Am prezentat anterior în material funcțiile `strcpy`, `strcat` și `strcmp` utile, respectiv pentru: copierea unui șir în alt șir, concatenarea unui șir la altul, compararea lexicografică a două șiruri. Funcțiile din titlul acestui paragraf sunt asemănătoare și, ce trebuie să știm despre ele pe scurt, este:

- Au un al treilea parametru de tip întreg, să îl numim `n`
- `strncpy` copiază în primul șir maxim `n` caractere (dacă al doilea șir are mai mult de `n` caractere)
- `strncat` concatenează la primul șir maxim `n` caractere (dacă al doilea șir are mai mult de `n` caractere)

- `strncmp` compară conținuturile luând în calcul doar primele `n` caractere (dacă ambele șiruri au cel puțin `n` caractere)

Exemple

`char s[101], t[101];`
`strcpy(t, "romania");`
`strncpy(s, t, 2);`
`cout<<s;`
	Se afișează `ro`

`char s[101], t[101];`
`strcpy(s, "pi");`
`strcpy(t, "romania");`
`strncat(s, t, 5);`
`cout<<s;`
	Se afișează `piroman`

`strncmp("marca", "mascat", 2);`	Rezultatul apelului este `0`. Compararea se face doar pe primele două poziții și acolo cele două cuvinte sunt identice.

Revenim la problema de mai sus care cerea să afișăm toate pozițiile la care un șir `c` apare ca secvență în altul, `s`. Oferim următoarea soluție. Cu un `for`, parcurgem toate pozițiile de început posibile ale celui de-al doilea șir. La o poziție fixată `i`, avem potrivide dacă `strncmp(s+i, c, strlen(c)) == 0`. Adică vom compara de la adresa respectivă din primul șir, cu al doilea șir, luând în calcul doar atâtea caractere cât este lungimea șirului de căutat.

```
#include <iostream>
#include <cstring>
using namespace std;
char s[101], *p, *q, c[101];
int main() {
    cin>>s>>c;
    for (int i=0;i+strlen(c) <= strlen(s);i++)
        if (strncmp(s+i, c, strlen(c)) == 0)
            cout<<i<<" ";
    return 0;
}
```

Citirea șirurilor de caractere de pe mai multe rânduri, inclusiv în cazul în care acestea conțin spații.

Nu am folosit încă mai sus citiri din fișiere dar nu este nimic special de spus. Se citește din fișiere după aceleași principii ca de la tastatură și ca la celelalte tipuri de date. De asemenea, la afișare.

În multe dintre exemplele din acest paragraf vom prelua date din fișiere pentru că este mai ușor de urmărit.

Citirea unui șir cu cin preia în șir doar până la primul caracter alb, deci nu și spațiile. Pentru a prelua spații soluția este să folosim cin.getline(sir, dimensiune), așa cum am spus mai sus.

Un apel `cin.getline` citește de la locul în care cursorul imaginar a fost lăsat în fișier de citirile anterioare, până la finalul rândului, mutând cursorul imaginar la începutul rândului următor și stocând în șirul dat ca parametru caractere până la finalul liniei. Acest lucru se întâmplă dacă parametrul dimensiune este mai mare decât numărul de caractere rămase pe linie. Se recomandă apelarea funcției cu valoare mare a lui dimensiune.

Exemple

`#include <fstream>`
`#include <iostream>`
`#include <cstring>`
`using namespace std;`
`ifstream fin("date.in");`
`char s[101], t[101];`
`int main() {`
`  fin.getline(s, 10);`
`  fin.getline(t, 10);`
`  cout<<s<<"\n"<<t;`
`  return 0;`
`}`
	Conținutul fișierului `date.in`
	`abcdefgh`
	`12345678`
	Se afișează pe ecran
	`abcdefgh`
	`12345678`

`ifstream fin("date.in");`
`char s[101], t[101];`
`int n;`
`int main() {`
`  fin>>n;`
`  fin.getline(s, 10);`
`  fin.getline(t, 10);`
`  cout<<s<<"\n"<<t;`
`  return 0;`
`}`
	Conținutul fișierului `date.in`
	`2`
	`abcdefgh`
	`12345678`
	Se afișează pe ecran
	Un rând liber iar pe al doilea `abcdefgh`
	Iată justificarea, și înțelegerea sa are mare importanță: după preluarea lui `n`, curorul imaginar din fișier rămâne la finalul primei linii (cea care îl conține pe `n`). Astfel, primul apel fin.getline preia șirul vid și abia al doilea apel citește efectiv simboluri, dar, cursorul fiind acum la începutul liniei a doua, se preia `abcdefgh`.

`fin>>n;`
`fin.get();`
`fin.getline(s, 10);`
`fin.getline(t, 10);`
`cout<<s<<"\n"<<t;`
	În contextul exemplului de mai sus, am adăugat linia scrisă îngroșat. Un astfel de apel, `fin.get`, fără parametri, are ca efect sărirea peste enter de la finalul rândului curent dacă acolo se află cursorul. Reținem acest mod de a trece pe rând nou avem întâi de citit un număr și apoi mai multe propoziții.

Un apel `fin.get(sir, dimensiune)` este similar cu `fin.getline`, cu observația că nu se trece pe rândul următor.

În continuare sunt prezentate două modalități de a prelua mai multe propoziții dintr-un fișier.

Pe prima linie a fișierului se află un număr `n` apoi, pe fiecare dintre următoarele `n` linii se află câte o propoziție (să spunem că este formată din litere mici, litere mari și spații).

În ambele exemple afișăm pe ecran propozițiile câte una pe rând, așa cum apar în fișier.

```
#include <fstream>
#include <iostream>
#include <cstring>
using namespace std;
ifstream fin("date.in");
char s[101];
int n, i;
int main() {
    fin>>n;
    fin.get();
    for (i=1;i<=n;i++) {
        fin.getline(s, 101);
        cout<<s<<"\n";
```

```
    }
    return 0;
}
```

```
#include <fstream>
#include <iostream>
#include <cstring>
using namespace std;
ifstream fin("date.in");
char s[101], t[101];
int n, i;
int main() {
    fin>>n;
    for (i=1;i<=n;i++) {
        fin.get();
        fin.get(s, 101);
        cout<<s<<"\n";
    }
    return 0;
}
```

La al doilea exemplu, se observă nevoia de a pune apelul de preluare a lui enter în interiorul structurii repetitive deoarece preluarea șirului se face cu `get` și nu cu `getline`, deci niciodată cursorul nu ar fi dus pe rândul următor, nici după preluarea la început a lui `n` și nici după.

Problemă rezolvată

Să se scrie un program care citește mai multe propoziții și determină propoziția de lungime maximă. Fișierul de intrare `lgmax.in` conține pe prima linie un număr natural `n`, iar pe următoarele `n` linii câte o propoziție alcătuită din litere ale alfabetului englez și spații. Fișierul de ieșire `lgmax.out` va conține pe prima linie propoziția de lungime maximă. Se dau maxim `100` de propoziții și fiecare cu maxim `255` de caractere. Dacă sunt mai multe propoziții de lungime maximă se va afișa prima. (pbinfo.ro, `#87`).

Exemplu

Date de intrare	Date de ieșire
4 somnoroase pasarele pe la cuiburi se aduna se ascund in ramurele noapte buna	pe la cuiburi se aduna

Rezolvare

```
#include <fstream>
#include <cstring>
using namespace std;
char s[260], sMaxim[260];
ifstream fin ("lgmax.in");
ofstream fout("lgmax.out");
int n, i, maxim;
int main () {
    fin>>n;
    fin.get();
    for (i=1;i<=n;i++) {
        fin.getline(s, 260);
        if (maxim < strlen(s)) {
```

```
            maxim = strlen(s);
            strcpy(sMaxim, s);
        }
    }
    fout<<sMaxim;
    return 0;
}
```

Vectori de cuvinte

Se pune problema stocării mai multor cuvinte. Noi știm că dacă dorim să memorăm un șir de valori de un anume tip putem folosi un vector. Aici ar trebui un vector de șiruri de caractere adică un vector de vectori de `char`. Limbajul ne pune simplu la dispoziție această facilitate, prin folosirea matricelor cu elemente de tip `char`.

Fie declarația

```
char a[20][30];
```

Efectiv avem la dispoziție memorie pentru `20` de șiruri de caractere și fiecare cu până la `29` de caractere utile (trebuie păstrată o poziție și pentru `NULL` de la finalul fiecărui șir).

Asta înseamnă că putem folosi pe a cu un singur indice, adică scriind `a[2]` ne referim la un singur șir de caractere, cel de pe linia cu indice `2` a matricei (a treia deci, știind că indexarea este de la `0`). Limbajul permite astfel de accesări.

Probleme rezolvate

1. Următorul program cere să introducem de la tastatură mai multe propoziții și apoi să le afișăm pe toate în ordine alfabetică.

Exemplu

Date de intrare	Date de ieșire
8 Fiind baiet paduri cutreieram Si ma culcam ades langa isvor, Iar bratul drept sub cap eu mi-l puneam S-aud cum apa suna-ncetisor: Un freamat lin trecea din ram in ram Si un miros venea adormitor. Astfel ades eu nopti intregi am mas, Bland inganat de-al valurilor glas.	Astfel ades eu nopti intregi am mas, Bland inganat de-al valurilor glas. Fiind baiet paduri cutreieram Iar bratul drept sub cap eu mi-l puneam S-aud cum apa suna-ncetisor: Si ma culcam ades langa isvor, Si un miros venea adormitor. Un freamat lin trecea din ram in ram

Rezolvare

```
#include <fstream>
#include <iostream>
#include <cstring>
using namespace std;
ifstream fin("date.in");
char a[20][50], aux[50];
int n, i, j;
int main() {
    fin>>n;
    for (i=1;i<=n;i++) {
        fin.get();
        fin.get(a[i], 101);
```

```
    }
    for (i=1;i<n;i++)
        for (j=i+1;j<=n;j++)
            if (strcmp(a[i], a[j]) > 0) {
                strcpy(aux, a[i]);
                strcpy(a[i], a[j]);
                strcpy(a[j], aux);
            }
    for (i=1;i<=n;i++)
        cout<<a[i]<<"\n";
    return 0;
}
```

Pe schema unui algoritm de sortare, ne referim la cate o linie a matricei deodata (vedem cum funcțiile de bibliotecă acceptă sintaxa) când vrem să indicăm unul dintre șirurile de caractere din vector.

Compararea se face lexicografic, deci utilizăm `strcmp` și la interschimbare ne trebuie o zonă auxiliară de memorie, așa cum am arătat si mai sus în material.

2. Mai sus în material am prezentat o problem ce avea la intrare un șir de caractere format doar cu simbolurile `7` și `9` și care înlocuia fiecare cifră cu cea "romană" corespunzătoare. Acum dorim să generalizăm acea problemă, adică șirul poate conține orice cifră de la `1` la `9`.

Exemplu

Date de intrare	Date de ieșire
`2974`	`IIIXVIIIV`

Rezolvare

Păstrăm principiul de la rezolvarea problemei amintite, formând rezultatul într-un șir.

Pentru a evita tratarea a `9` cazuri diferite vom folosi un vector de constante în care pe poziția `i` este șirul ce reprezintă cifra romană `i`. Acesta este un vector de șiruri de caractere, deci o matrice și urmăriți în cod modul de inițializare și de folosire a sa. Am inițializat primul element cu șirul vid pentru că cifra `0` nu apare la intrare și remarcați trecerea de la incicele `i` la simbolul cifră al `i`-lea.

```
#include <iostream>
#include <cstring>
using namespace std;
char s[21], t[101];
char rom[10][5] = {"","I","II","III","IV","V","VI","VII","VIII","IX"};
int main () {
    cin>>s;
    strcpy(t, ""); /// sau t[0] = 0;
    for (int i=0;s[i]!=0;i++)
        strcat(t, rom[s[i]-'0']);
    strcpy(s, t);
    cout<<t;
}
```

Funcția strtok

Sunt dese situațiile când apare cerința de a extrage cuvintele dintr-o propoziție. Obținem acest lucru ușor cu funcția amintită în subtitlu.

Sufixul "`tok`" vinde de la `token`, așadar funcția permite, în general, descompunerea unui șir mai mare în tokeni – bucăți mai mici – având precizat care sunt caracterele considerate separatori.

Sintaxa:

```
strtok(sir de despartit, sir de separatori)
```

Ambii parametri ai funcției sunt pointeri la char iar rezultatul este tot pointer la `char`.

Pentru a putea prezenta mai ușor efectul unui apel ne vom folosi de un exemplu.

Să presupunem că avem șirul s: `- azi,  miercuri, este foarte  cald.` (considerăm – ca fiind primul caracter si `.` ca fiind ultumul).

Facem apelul `char *p = strtok(s, "-, .");`

Șirul de delimitatori este format din liniuță, virgulă, spațiu și punct. Prin acesta dorim să izolăm primul cuvânt considerând că între ghilimele sunt separatorii: se sare peste toți separatorii începând cu pointerul dat prin primul parametru până se ajunge la `a`, care este primul caracter neseparator. Se notează această adresă, pentru că se va returna de către funcție, apoi se sare peste toate literele (adică neseparatori) până la primul separator sau la final. Aici se va ajunge la virgula de după `azi`. Funcția pune acolo `NULL`.

Acum, o afișare alui `p` ar avea ca efect tipărirea `azi`, adică de la adresa returnată până în primul loc unde se află `NULL`. Așadar, funcția extrage tokenul returnând de fapt adresa unde el începe în șirul din care face parte și punând `NULL` după el. Deci mai tragem și concluzia că apelurile `strtok` modifică șirul pe care se aplică.

Cum extragem următorul token ? Poate pare puțin ciudat acum, dar, după primul apel făcut ca mai sus, următoarele se fac cu `NULL` în dreptul primului parametru. Lucrurile funcționează la fel ca la primul apel.

Considrăm codul

```
p = strtok(s, "-, .");
cout<<p<<"\n";
p = strtok(NULL, "-, .");
cout<<p<<"\n";
p = strtok(NULL, "-, .");
cout<<p<<"\n";
```

Aplicat asupra șirului de mai sus acesta are efectul:

Șirul inițial:	`- azi,  miercuri, este foarte  cald.`

Ecran:	`azi`
	`miercuri`
	`este`

Șirul la final	`- azi#  miercuri# este#foarte  cald.`

Prin # am simbolizat `NULL` (Asta doar pentru a fi mai ușor de prezentat, dar evident că # este caracter separat, cu propriul său cod ASCII.

Dacă am mai fi apelat de două strtok am fi obținut și celelalte cuvinte. Funcția `strtok` returnează `NULL` când la un apel nu mai găsește niciun token, așadar putem folosi asta ca și condiție de oprire.

Codul standard cu care extragem cuvintele dintr-un șir este:

```
char s[101], delim[] = "simbolurile delimitatorilor", *p;
p = strtok(s, delim);
while (p!=NULL) {
    cout<<p<<" ";
    p = strtok(null, delim);
}
```

Probleme rezolvate

1. Dintr-un fișier se citește o propoziție de pe primul rând. Considerăm că sunt date doar pe primul rând și acestea pot fi litere mici ale alfabetului și spații. Spațiile sunt separatori de cuvinte. Să se afișeze

cuvintele pe un singur rând, separate prin câte un spațiu, în ordine inversă a apariției lor. Considerăm șirul dat de maxim `100` de caractere și lungimea maximă a unui cuvânt de `20` de caractere.

Exemplu

Date de intrare	Date de ieșire
`o zi frumoasa de vacanta`	`vacanta de frumoasa zi o`

Rezolvare

Prezentăm mai multe soluții la această problemă.

Soluția 1

Citim propoziția într-un șir de caractere, separăm cuvintele cu `strtok` și le stocăm într-o matrice (vector de cuvinte), afișându-le apoi invers.

```
#include <fstream>
#include <iostream>
#include <cstring>
using namespace std;
char s[101], a[101][21], *p;
int n;
ifstream fin("date.in");
int main () {
    fin.getline(s, 101);
    p = strtok(s, " ");
    while(p!=NULL) {
        n++;
        strcpy(a[n], p);
        p = strtok(NULL, " ");
    }
    for (int i=n;i>=1;i--)
        cout<<a[i]<<" ";
}
```

Solutia 2

Aceasta problema are o particularitate, separatorii sunt doar spațiile și noi putem profita de asta ținând cont că o citire cu cin simplu citește doar până la spațiu.

Așadar:

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;
char s[21], a[101][21];
int n;
ifstream fin("date.in");
int main () {
    while(fin>>s) {
        n++;
        strcpy(a[n], s);
    }
    for (int i=n;i>=1;i--)
```

```
        cout<<a[i]<<" ";
}
```

Soluția 3

Aici ne vom folosi de recursivitate și vom declara un șir local în care citim, făcând afișările la revenire. Nu mai este nevoie să declarăm matricea dar asta nu înseamnă că pe cazul defavorabil facem exonomie de memorie (la fiecare nivel din stivă e câte un șir nou alocat).

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;
int n;
ifstream fin("date.in");
void citire() {
    char s[21];
    if(fin>>s) {
        citire();
        cout<<s<<" ";
    }
}
int main () {
    citire();
}
```

2. Se dă o propoziție care conține numai litere mici ale alfabetului englez și spații. Să se afișeze cuvintele din propoziție care conțin numai vocale. (pbinfo.ro, #973)

Rezolvare

Se separă cuvintele folosind strtok iar verificarea fiecărui cuvânt în parte se face tratându-l ca pe un vector și parcurgându-l caracter cu caracter.

```
#include <iostream>
#include <cstring>
using namespace std;
char *p;
char s[256];
int ok,i;
int main(){
    cin.get(s,256);
    p=strtok(s," ");
    while(p!=NULL){
        ok=1;
        for(i=0;i<strlen(p);i++)
            if(strchr("aeiou",p[i])==0)
                ok=0;
        if(ok)
            cout<<p<<"\n";
        p=strtok(NULL," ");
    }
    return 0;
}
```

3. Se dă o propoziție formată din litere mici ale alfabetului englez, spații și semnele de punctuație virgulă și punct. Determinați un cuvânt palindrom din propoziție, primul în ordine alfabetică (pbinfo.ro, #975).

Exemplu

Date de intrare	Date de ieșire
ele deschid un capac, aerisirea este ok	aerisirea

Rezolvare

Separăm cuvintele cu strtok și folosim o funcție care face verificarea la nivel de element. De remarcat modul ales de a inițializa variabila în care rămâne soluția. Noi avem de determinat un "cuvânt minim", deci trebuie făcută inițializarea cu ceva mare. Așadar, punem pe prima poziție z, adică ultimul caracter apoi incrementăm, aigurându-ne că ajundu-se la o valoare mai mare ca a oricărui cuvânt.

```
#include <iostream>
#include <cstring>
using namespace std;
int n, k, i, j;
char s[302], *p;
char sep[] = " ,.";
char voc[] = "aeiou";
int estePalindrom(char s[]) {
    int st = 0, dr = strlen(s)-1;
    while (st < dr) {
        if (s[st]!=s[dr])
            return 0;
        st++;
        dr--;
    }
    return 1;
}
char sol[12];
int main () {
    cin.get(s, 301);
    strcpy(sol,"z");
    sol[0]++;
    p = strtok(s, sep);
    while (p!=0) {
        if (estePalindrom(p)) {
            if (strcmp(p, sol) < 0)
                strcpy(sol, p);
        }
        p = strtok(NULL, sep);
    }
    if (sol[0] <= 'z') {
        cout<<sol;
    } else {
        cout<<"IMPOSIBIL";
    }
    return 0;
}
```