Facultatea de Matematică și Informatică
Lecții de pregătire – Admitere 2012

Șiruri de caractere

Radu Ionescu
raducu.ionescu@gmail.com

Ruxandra Marinescu
verman@fmi.unibuc.ro

- Un caracter – memorat prin utilizarea codului ASCII

C/C++

```
char c='a';

int x;

x=c; //x=(int)c;

c=x+3;//c=(char)(x+3);

printf("%d %c %d",x,c,c-'a');
```

Pascal

```
var c:char; x:integer ;

c:='a';

x:=ord(c);

c:=chr(x+3); {char(x+3);}

writeln(x,' ',c, ' ',
        ord(c)-ord('a'));
```

- Un caracter – memorat prin utilizarea codului ASCII

C/C++

```
char c='a';

int x;

x=c; //x=(int)c;

c=x+3;//c=(char)(x+3);

printf("%d %c %d",x,c,c-'a');
```

97 d 3

Pascal

```
var c:char; x:integer ;

c:='a';

x:=ord(c);

c:=chr(x+3); {char(x+3);}

writeln(x,' ',c, ' ',
        ord(c)-ord('a'));
```

97 d 3

- **Șiruri de caractere:**

Declarare, inițializare, afișare

C/C++

```
#include<stdio.h>
//#include<iostream.h>
int main(){
      char c[10]="cuvant";
      //char c[]="cuvant";
      printf("%s",c);
      //cout<<c;
}
```

Pascal

```
var c:string;
{var c:string[10];}
begin
      c:='cuvant';
      writeln(c);
end.
```

Memorare

C/C++

0	1	2	3	4	5	6
c	u	v	a	n	t	\0

Pascal

0	1	2	3	4	5	6
chr(6)	c	u	v	a	n	t

↑ **Lungime**

Citire, accesarea unui caracter

C/C++

```
int main(){
        char c[10];
        scanf("%s",c);
        c[0]='C';
        printf("%s",c);
}
```

Pascal

```
var c:string[10];
begin
        readln(c);
        c[1]:='C';
        writeln(c);
end.
```

Funcții. Operații asupra șirurilor – C/C++

- `char c[10]="cuvant", *c1;`

  `c` – adresa vectorului (primului element, octet)

  – adresă **constantă**

  `c1` – adresă (variabilă)

  - `c1 = c;  //DA`
  - `c = c1;  //NU`

- Operații

  ```
  c1 = c+3;
  printf("%s",c1-1);   ⟶   vant
  ```

Funcții. Operații asupra șirurilor – C/C++

- `#include<string.h>`
- `char c[10]="cuvant";`
- **Lungime**
  - `size_t strlen(char* s);`
    - `n=strlen(c);` ⟶ `n=6`
- **Copiere**
  - `char* strcpy(char* dest, char* sursa);`
    - `char *c1="cuvantul";`
      `strcpy(c,c1);` ⟶ `c devine "cuvantul"`

Funcții. Operații asupra șirurilor – C/C++

- `char c[10]="cuvant";`
- **Concatenare**
  - `char*` **`strcat`**`(char* dest, char* sursa);`
  - `char*` **`strncat`**`(char* dest, char* sursa, size_t nr);`
    - `char *c1="ul";`
      `strcat``(c,c1);` ⟶ `c devine "cuvantul"`
    - `char *c1="ultim";`
      `strncat``(c,c1,2);` ⟶ `c devine "cuvantul"`
- **Comparare**
  - `int` **`strcmp`**`(const char* c1, const char* c2);`

$> 0 \Leftrightarrow c1 > c2$

$= 0 \Leftrightarrow c1 = c2$

$< 0 \Leftrightarrow c1 < c2$

Funcții. Operații asupra șirurilor – Pascal

- Presupunem `c:='cuvant'`
- **Lungime**
  - `function` **`length`**`(s: string):integer;`
    - `n:=length(c);` ⟶ `n:=6`
- **Copiere**
  - `c1:=c;`
  - function **`copy`**(s:string; poz,lung:integer):string;
    - `c1:=copy(c,2,4);` ⟶ `c1:='uvan'`
- **Concatenare**
  - `c2:=c+c1;` ⟶ `c2:='cuvantuvan'`
- **Comparare**
  - `c1=c2` `c1>c2` `c1<c2`

Funcții. Operații asupra șirurilor – C/C++

- `char c[10]="cuvant";`
- **Căutare**
  - `char*` **`strchr`**(const char* s, int c);
    - `char *c1=strchr(c,'a');`
      `printf("%s\n",c1);` ⟶ `ant`
      `printf("pozitia %d",c1-c);` ⟶ `3`
  - `char*` **`strstr`**(const char* sir, const char* subsir);
    - `char *c1=strstr(c,"an");`
      `printf("%s\n",c1);` ⟶ `ant`
      `printf("pozitia %d",c1-c);` ⟶ `3`
- **Ștergere**
  - `strcpy(c+1,c+3);` ⟶ `c devine "cant"`

Funcții. Operații asupra șirurilor – Pascal

- **Căutare**

  function **pos**(subsir,sir:string):byte;

  - `n:=pos('uv',c);` ⟶ `n:=2;`

- **Inserare**

  procedure **insert**(sir_de_ins:string;

  var sir_unde_ins:string; poz:integer);

  - `insert('ne',c,1);` ⟶ `c devine 'necuvant'`

- **Ștergere**

  procedure **delete**(var sir:string;

  poz,lung:integer);

  - `delete(c,1,4);` ⟶ `c devine 'vant'`

Vectori de frecvențe

- Să presupunem că avem un cuvânt format doar cu litere mici. Putem asocia acestui cuvânt un vector de frecvente (de apariții) cu semnificația

```
nr[i]= de cate ori apare a i-a litera din
       alfabet în cuvânt
```

Scrieți o funcție care primește ca parametru un șir de caractere care conține numai litere mici și returnează litera cu cele mai multe apariții

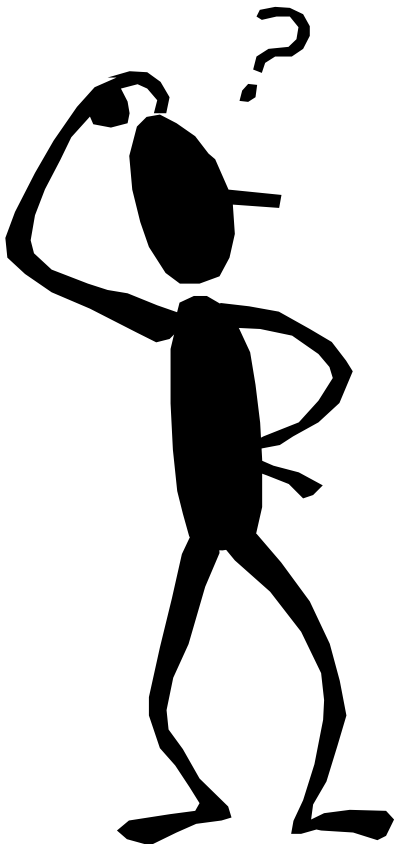
```
char apar(char s[]){
  int i,nr[30],pmax;

  pmax=0;
  for(i=0;i<='z'-'a';i++)
     nr[i]=0;
  for(i=0;i<strlen(s);i++)
     nr[s[i]-'a']++;
  for(i=1;i<='z'-'a';i++)
     if(nr[i]>nr[pmax])
        pmax=i;

  return pmax+'a';
}
```

```
function apar(s:string):char;
var  i:integer; ic,pmax:char;
     nr:array['a'..'z'] of integer;
begin
    pmax:='a';
    for ic:='a' to 'z' do
        nr[ic]:=0;
    for i:=1 to length(s) do
        nr[s[i]]:=nr[s[i]]+1;
    for ic:='b' to 'z' do
        if nr[ic]>nr[pmax] then
            pmax:=ic;

    apar:=pmax;
end;
```

Scrieți o funcție care primește ca parametru un șir de caractere și ordonează crescător literele acestuia

```
void sort(char s[]){

  int i,nr[30],k,j;

  for(i=0;i<='z'-'a';i++)
      nr[i]=0;

  for(i=0;i<strlen(s);i++)
      nr[s[i]-'a']++;

  k=0;
  for(i=0;i<='z'-'a';i++)
     for(j=1;j<=nr[i];j++){
        s[k]=i+'a';
        k++;
     }
}
```

```
procedure sort(var s:string);
var  i,k,j:integer; ic:char;
     nr:array['a'..'z'] of integer;
begin
    for ic:='a' to 'z' do
        nr[ic]:=0;

    for i:=1 to length(s) do
        nr[s[i]]:=nr[s[i]]+1;

    k:=1;
    for ic:='a' to 'z' do
        for j:=1 to  nr[ic] do
        begin
           s[k]:=ic;
           k:=k+1;
        end;

end;
```



Probleme

Scrieți o funcție care primește ca parametru un șir de caractere care conține numai litere mici și returnează numărul de consoane ale acestuia

```c
int nr_cons(char c[]){

 char *voc;
 int n,i,nr;

 voc="aeiou";
 n=strlen(c);
 nr=0;
 for(i=0;i<n;i++)
   if(strchr(voc,c[i])==0)
        nr++;
  return nr;
}
```

```
function nr_cons(c:string)
                          :integer;
var voc:string;
     n,nr,i:integer;
begin
    voc:='aeiou';
    n:=length(c);
    nr:=0;
    for i:=1 to n do
      if pos(c[i],voc)=0 then
         nr:=nr+1;
    nr_cons:=nr;
end;
```

Scrieți o funcție care primește ca parametru două șiruri și șterge aparițiile primului șir în cel de al doilea

```
void stergere(char subs[],
              char s[]){
    char *a;
    int n=strlen(subs);
    a=strstr(s,subs);
    while(a){
        strcpy(a,a+n);
        a=strstr(a,subs);
    }
}
```

```
procedure stergere(subs:string;
                   var s:string);
var a:string;
    n,poz:byte;
begin
    n:=length(subs);
    poz:=pos(subs,s);
    a:=s;
    s:='';
    while poz<>0 do
    begin
        s:=s+copy(a,1,poz-1);
        delete(a,1,poz+n-1);
        poz:=pos(subs,a);
    end;
    s:=s+a;
end;
```

Se citesc n cuvinte (de minim două litere). Fiecare cuvânt este format doar din litere mici. Spunem că două cuvinte rimează dacă au ultimele două caractere identice.

a) Scrieți o funcție care primește ca parametri două șiruri și testează dacă acestea rimează
b) Afișați cel mai mare grup de cuvinte citite care rimează.

Exemplu: Pentru cuvintele

```
care, masa, mare, casa, test, lucrare
```

se va afișa

```
care mare lucrare
```

a)

```
int rimeaza(char s1[],char s2[]){

//if ((s1[strlen(s1)-1]==
       s2[strlen(s2)-1])&&
      (s1[strlen(s1)-2]==
       s2[strlen(s2)-2]))

if(strcmp(s1+strlen(s1)-2,
          s2+strlen(s2)-2)==0)
        return 1;
    else
        return 0;
}
```

```
function rimeaza(s1,s2:string)
                        :boolean;
begin
{if (s1[length(s1)]=
     s2[length(s2)]) and
    (s1[length(s1)-1]=
     s2[length(s2)-1]) then}

if copy(s1,length(s1)-1,2)=
   copy(s2,length(s2)-1,2)
then

        rimeaza:=true
    else
        rimeaza:=false;
end;
```

b)

Varianta 1 – Numărăm pentru fiecare cuvânt cu câte cuvinte din șir rimează (mai exact cu câte cuvinte care sunt după el în șir rimează)

Varianta 1 – Pascal

```
var c:array[1..100] of string[50];
   i,j,imax,lungm,lung,n:integer;
begin
    readln(n);
    for i:=1 to n do readln(c[i]);
    lungm:=1; imax:=1;
    for i:=1 to n-1 do
    begin
        lung:=1;
        for j:=i+1 to n do
             if rimeaza(c[i],c[j]) then
                 lung:=lung+1;
        if lung>lungm then
        begin lungm:=lung;imax:=i;
        end;
    end;
    for i:=1 to n do
        if rimeaza(c[imax],c[i])  then
            write(c[i],' ');
end.
```

Varianta 1 – C

```
int n;
char c[100][50];
int main(){
    int i,j,imax,lungm,lung;
    scanf("%d",&n);
    for(i=0;i<=n-1;i++) scanf("%s",c[i]);
    lungm=1; imax=0;
    for(i=0;i<n-1;i++){
        lung=1;
        for(j=i+1;j<n;j++)
            if(rimeaza(c[i],c[j])) lung++;
            if(lung>lungm){ lungm=lung; imax=i;}
   }
    for(i=0;i<=n-1;i++)
        if(rimeaza(c[imax],c[i]))
            printf("%s ",c[i]);
}
```

b)

Varianta 2 – Ordonăm cuvintele crescător după ultimele două litere și determinăm cea mai lungă subsecvență de cuvinte (consecutive) care rimează în șirul obținut

Varianta 2 – Pascal

```
var i,poz,lung,pozm,lungm:integer;
begin
    readln(n);
    for i:=1 to n do readln(c[i]);
    quicksort(1,n); {dupa ultimele doua litere}
    poz:=1;pozm:=1;lung:=1;lungm:=1;
    for i:=2 to n do
        if rimeaza(c[i-1],c[i]) then lung:=lung+1
        else
        begin
             if lung>lungm then
             begin lungm:=lung;pozm:=poz;
             end;
             poz:=i;lung:=1;
        end;
    if lung>lungm then
    begin lungm:=lung;pozm:=poz;
    end;
    for i:=pozm to pozm+lungm-1 do  write(c[i],' ');
end.
```

```
function pozitie(p,u:integer):integer;
var i,j,depli,deplj,aux:integer; a:string;
begin
  i:=p;j:=u; depli:=0;deplj:=-1;
  while i<j do begin
    if copy(c[i],length(c[i])-1,2)>
       copy(c[j],length(c[j])-1,2) then begin
       a:=c[i]; c[i]:=c[j]; c[j]:=a;
       aux:=depli; depli:=-deplj; deplj:=-aux;
    end;
    i:=i+depli;j:=j+deplj;
  end;
  pozitie:= i;
end;
procedure quicksort(p,u:integer);
var k:integer;
begin
  if p<u then begin
     k:=pozitie(p,u); quicksort(p,k-1); quicksort(k+1,n);
  end;
end;
```

Varianta 2 – C

```
int main(){
    int i,poz,lung,pozm,lungm;
    scanf("%d",&n);
    for(i=0;i<=n-1;i++)
        scanf("%s",c[i]);
    quicksort(0,n-1); //dupa ultimele doua litere
    poz=0; pozm=0; lung=1; lungm=1;
    for(i=1;i<=n-1;i++)
        if(rimeaza(c[i-1],c[i])) lung++;
        else{
            if(lung>lungm) {lungm=lung; pozm=poz;}
            poz=i;lung=1;
        }
    if(lung>lungm) {lungm=lung; pozm=poz;}
    for(i=pozm;i<=pozm+lungm-1;i++)
        printf("%s ",c[i]);
}
```

```
int pozitie(int p,int u){
 int i=p,j=u, depli=0,deplj=-1,aux;
 while(i<j){
    if(strcmp(c[i]+strlen(c[i])-2,c[j]+strlen(c[j])-2)>0){
        char a[50];
        strcpy(a,c[i]); strcpy(c[i],c[j]); strcpy(c[j],a);
        aux=depli; depli=-deplj; deplj=-aux;
    }
    i+=depli;j+=deplj;
 }
 return i;
}

void quicksort(int p,int u){
     if (p<u){
         int k=pozitie(p,u);
         quicksort(p,k-1);  quicksort(k+1,n-1);
     }
}
```

b)

Varianta 3 – Folosim o matrice în care numărăm aparițiile fiecărui sufix de două litere:

nr[i][j] = de câte ori apare în șir sufixul format din literele i și j ale alfabetului

```
var n,i:integer; ic,jc,c1,c2,imax,jmax:char;
    c:array[1..100] of string[50]; suf:string[2];
    nr:array['a'..'z','a'..'z'] of integer;
begin
    readln(n); for i:=1 to n do readln(c[i]);
    for ic:='a' to 'z' do
        for jc:='a' to 'z' do
            nr[ic][jc]:=0;
    imax:='a';jmax:='a';
    for i:=1 to n do
    begin
        c1:=c[i][length(c[i])-1]; c2:=c[i][length(c[i])];
        nr[c1][c2]:=nr[c1][c2]+1;
        if nr[c1,c2]>nr[imax,jmax] then begin
            imax:=c1; jmax:=c2;
        end;
    end;
    suf[1]:=imax;suf[2]:=jmax;suf[0]:=chr(2);{suf:=imax+jmax;}
    for i:=1 to n do
        if copy(c[i],length(c[i])-1,2)=suf then
            write(c[i],' ');
end.
```

Varianta 3 - C

```
int n,i,j,imax=0,jmax=0;
int nr[30][30];
char c[100][50],suf[3],c1,c2;
scanf("%d",&n);
for(i=0;i<=n-1;i++) scanf("%s",c[i]);
for(i=0;i<='z'-'a';i++)
    for(j=0;j<='z'-'a';j++)
         nr[i][j]=0;
for(i=0;i<=n-1;i++){
    c1=c[i][strlen(c[i])-2];
    c2=c[i][strlen(c[i])-1];
    nr[c1-'a'][c2-'a']++;
    if(nr[c1-'a'][c2-'a']>nr[imax][jmax]){
        imax=c1-'a'; jmax=c2-'a';
    }
}
suf[0]=imax+'a'; suf[1]=jmax+'a'; suf[2]=0;
for(i=0;i<=n-1;i++)
    if(strcmp(c[i]+strlen(c[i])-2,suf)==0)
        printf("%s ",c[i]);
```

Dat un șir de paranteze rotunde, să se determine dacă acesta este corect

```
int corect(char c[100]){
    int dif,n,ok,i;

    n=strlen(c);
    dif=0;  ok=1; i=0;
    while((i<n) && ok){
      if(c[i]=='('){
         dif++;
         if(dif>n-1-i)
           ok=0;
      }
      else{
         dif--;
         if (dif<0)ok=0;
      }
      i++;
    }
    //if(dif!=0) ok=0;
    return ok;
}
```

```
function corect(c:string):boolean;
var dif,n,i:integer; ok:boolean ;
begin
   n:=length(c);
   dif:=0;  ok:=true; i:=1;
   while(i<=n) and ok do begin
      if c[i]='(' then begin
          dif:=dif+1;
          if dif>n-i then ok:=false;
      end
      else begin
          dif:=dif-1;
          if dif<0 then ok:=false;
      end;
      i:=i+1;
  end;
  {if dif<>0 then ok:=false;}
  corect:=ok;
end;
```

Dat un număr n, să se genereze toate șirurile corecte de n paranteze rotunde

```
int n; char s[100];
```

```
var n:byte;  s:string[100];
```

```
void gen(int i,int dif){


    if(i==n) {
        s[n]=0;
        printf("%s\n",s);
    }
    else
       if(dif<n-i-1){
           s[i]='(';
           gen(i+1,dif+1);
       }
       if(dif>0){
           s[i]=')';
           gen(i+1,dif-1);
        }
}
```

```
procedure gen(i:byte;dif:byte);
begin
    if i=n+1 then begin
          s[0]:=chr(n);
          writeln(s);
    end
    else begin
       if dif<n-i then begin
            s[i]:='(';
            gen(i+1,dif+1);
        end;
       if dif>0 then begin
            s[i]:=')';
            gen(i+1,dif-1);
        end;
    end;
end;
```

```
 scanf("%d",&n);
 gen(0,0);
```

```
readln(n);
gen(1,0);
```

Dat un șir de paranteze rotunde și drepte, să se determine dacă acesta este corect

```
char pereche(char c){
     if (c=='(')
        return ')';
     else
        return ']';
}
```

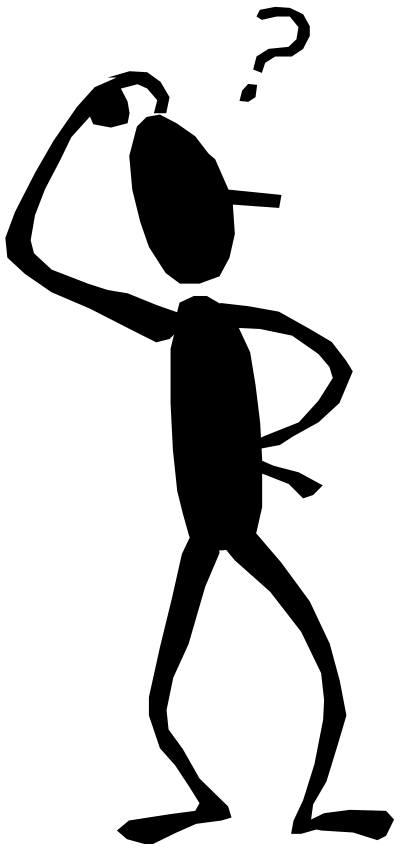
```
function pereche(c:char):char;
begin
     if c='(' then
        pereche:= ')'
     else
        pereche:=']';
end;
```

Varianta Pascal

```
function corect(c:string):boolean;
var n,i,k:integer; ok:boolean ;  st:string;
begin
   n:=length(c);
   k:=0;  ok:=true; i:=1;
   while(i<=n) and ok do  begin
    if (c[i]='(') or (c[i]='[') then begin
          k:=k+1;
          st[k]:=c[i];
    end
    else begin
          if k=0 then ok:=false;
          if c[i]<>pereche(st[k]) then ok:=false;
          k:=k-1;
    end;
    i:=i+1;
  end;
  if k>0 then  ok:=false;
  corect:=ok;
end;
```

Varianta C

```
int corect(char c[100]){
    int st[100],n,ok,i,k;
    n=strlen(c);
    k=-1;  ok=1; i=0;
    while((i<n) && ok){
        if((c[i]=='(') || (c[i]=='[') ){
             k++;
             st[k]=c[i];
        }
        else{
            if(k==-1) ok=0;
            if (c[i]!=pereche(st[k])) ok=0;
            k--;
        }
        i++;
    }
    if(k>=0) ok=0;
    return ok;
}
```



Probleme propuse

a) Scrieți o funcție care primește ca parametri două cuvinte și verifică dacă unul este permutare circulară a celuilalt

b) Se citește un șir de n cuvinte formate din litere mici, fiecare cuvânt având litere distincte. Determinați numărul maxim de cuvinte din șir care sunt permutări circulare ale aceluiași cuvânt.

Exemplu: Pentru cuvintele

```
care, mar, arec, arm, reca, alt
```

se va afișa

```
3
```

b)

Idee: Permutăm fiecare cuvânt la stânga astfel încât cea mai mică literă a sa să ajungă pe prima poziție. Problema se reduce astfel la determinarea cuvântului care apare de cele mai multe ori în șir. Pentru a determina acest cuvânt ordonăm lexicografic șirul de cuvinte și determinăm subsecvența de lungime maximă formată cu cuvinte egale.

O mulțime de anagrame este o mulțime de cuvinte cu proprietatea că oricare două cuvinte din mulțime au aceleași litere (sunt anagrame)

Dat un șir de n cuvinte, să se afișeze care este numărul maxim de elemente ale unei mulțimi de anagrame din acest șir

Exemplu:

Pentru șirul

este, mare, sete, marea, arme, eram

numărul maxim este 3, pentru mulțimea

{mare, arme, eram}

Se dă un șir de n cuvinte. Să se afișeze cuvintele distincte care apar în șir, în ordinea lungimii lor