Pointeri. Şiruri decaractere.

Un ***pointer*** este o dată care are ca valori adrese de memorie.
Pointerii sunt utilizaţi în general:

- în lucrul cu tablouri
- la transmiterea parametrilor în funcţii
- pentru accesarea directă a memoriei
- la alocarea dinamică a memoriei

Adresa unei variabile este adresa primului octet din reprezentarea internă a variabilei respective. Variabilele de tip pointer au ca valoare adrese de memorie.

Obs.1. Limbajul C++ face distincţie între natura adreselor care pot fi memorate.
2. Nu trebuie confundată adresa unei variabile cu conţinutul variabilei respective.

Declararea unui pointer de date, are următorul format:

```
tip* var_pointer;        // tip reprezintă tipul de bază al pointerului şi va indica tipul
                            //datei memorate

ex:  int *p1, *p2;       //p1, p2 sunt pointeri la întregi, deci reţin adresele unor zone //de
                            memorie la care vor fi memorate date întregi
     char *c;            // c va indica o zonă de memorie la care va fi memorat un
                            //caracter
```

Obs. 3.Tipul de bază al pointerului poate fi orice tip al limbajului inclusiv tipul **void**.
Un pointer cu tipul de bază **void*** se numeşte **pointer generic.**
4.Există o constantă pointer specială, denumită NULL, cu semnificaţia "pointerul nu conţine adresa nici unei zone de memorie"; valoarea acestei constante este 0.

Operaţii cu pointeri

1. Operaţia de referenţiere

-este operaţia prin care putem obţine adresa de memorie a unei variabile
-adresa unei variabile se poate obţine cu ajutorul operatorului unar **&** care va precede numele variabilei; rezultatul acestei operaţii este un pointer

```
ex:  int *p,a;
     p=& a;          // p va reţine adresa lui a
```

2. Operaţia de dereferenţiere

-este operaţia prin care putem accesa conţinutul memorat la o anumită adresă
-operaţia se realizează folosind operatorul unar ***** care va precede numele variabilei

```
ex:  int *p,a;
     p=& a;          // p va reţine adresa lui a
     *p=3;
     cout<<a;      //va afişa pe ecran valoarea 3
```

3. Operaţia de incrementare/decrementare

-are ca efect indicarea elementului următor/anterior celui indicat de pointerul iniţial

-se creşte/micşorează adresa memorată în pointer, cu numărul de octeţi necesari pentru a memora o dată de tipul de bază al pointerului **sizeof(tip)**

ex: long int *p;
```
p++;            // adresa memorată de p va creşte cu 4 octeţi
p--;            //adresa memorată de p se micşorează cu 4 octeţi
```

4. Adunarea / scăderea dintre un pointer şi un întreg p+n / p-n

- are următorul efect: adresa memorată în pointer creşte/se micşorează cu **n*sizeof(tip)**, unde tip reprezintă tipul de bază al pointerului p

5. Scăderea a doi pointeri

-rezultatul obţinut are valoare întreagă şi indică numărul de elemente dintre cei doi pointeri

6. Compararea a doi pointeri

-asupra pointerilor care au acelaşi tip de bază se pot aplica operatorii relaţionali şi de egalitate

ex:
```
int *p1,*p2,a;
a=2;
p1=&a;
p2=p1;
if (p1==p2) cout<<"indică aceeaşi zonă de memorie";
```

7. Afişarea unui pointer

în limbajul C

-se utilizează funcţia printf cu specificatorul de format %p

```
int *r,a;
a=2;
r=&a;
printf("%p",r);
```

în limbajul C++

-se utilizează operatorul de afişare

```
cout<<r;
```

Legătura dintre pointeri şi tablouri

Numele unui tablou este un pointer constant care are ca valoare adresa primului element din tablou

Utilizarea pointerilor reprezintă una din posibilităţile de accesare a elemenelor unui tablou.

ex: tip x[100]; tip este tipul de bază al elementelor vectorului

		expresii echivalente
x	&x[0]	adresa primului element din vector
x+i	&x[i]	adresa elementului de pe poyi'ia i din vector
*x	x[0]	primul element din vector
*(x+i)	x[i]	elementul de pe poziţia i din tablou

Şirurile de caractere

Un şir de caractere este o structură de date formată dintr-o succesiune de caractere care se termină cu caracterul NULL (' \0').

Implementarea şirurilor de caractere se face sub forma unui tablou unidimensional (vector) ale cărui elemente sunt de tip caracter, fiecare caracter fiind reprezentat prin codul său ASCII.

Şirul de caractere se deosebeşte de vectorii cu alte tipuri de elemente prin marcatorul logic utilizat la sfârşitul şirului - NULL (care are codul ASCII 0).

Declararea

ex: char x[5]; // am declarat şirul x care va putea conţine maxim 5 caractere
cin>>x;

x	'm'	'a'	'm'	'a'	'\0'
	0	1	2	3	4

Dimensiunea vectorului de caractere trebuie declarată cu un caracter mai mult decât cel mai mare şir de caractere pe care îl poate conţine, pentru a exista loc şi pentru terminatorul şirului (caracterul '\0').

O constantă şir de caractere este o succesiune de caractere delimitată de ghilimele.

ex: char şir[256]="Buna ziua";

Citirea

în limbajul C++

-se poate utilize operatorul >> sau funcţiile get sau getline

```
char c, sir[201];

fin>>c;
fin.getline(sir, 200);
//sau fin.get(sir,200);fin.get();
```

în limbajul C

-se foloseşte specificatorul de format %c pentru caractere şi %s pentru şiruri

```
char c, sir[201];

fscanf(fin,"%c", &c);
fscanf(fin,"%s",sir);
```

Prototipul funcţiilor get şi getline sunt:

getline(char *s, int n, char c='\n');
get(char *s, int n, char c='\n');

Diferenţa dintre funcţiile **get** şi **getline** este faptul că funcţia getline preia din stream-ul de intrare şi delimitatorul, în timp ce funcţia get nu-l extrage.

Scrierea

în limbajul C++

```
fout<<c<<' '<<s;
```

în limbajul C

```
fprintf(fout," %c", c);
fprintf(fout," %s", sir);
puts(sir);
```

Obs.

1. Citirea şirurilor de caractere la care se utilizează operatorul **>>** se opreşte la întâlnirea primului caracter alb.

2.Dacă delimitatorul este altul decât '\n', acesta trebuie precizat pe poziţia corespunzătoare lui în funcţia getline.

ex:getline(sir,200,'!'); // sir va conţine caracterele citite până la întâlnirea caracterului ! sau pană la citirea a 200 de caractere.

3.Fişierul antet stdio.h conţine o funcţie specială pentru afişarea şirurilor de caractere. Funcţia afişează caracterele şirului până la NULL şi apoi afişează '\n'.
Prototipul ei este: int puts(const char* sir);

Prelucrarea şirurilor de caractere se poate face:

a) prin parcurgerea caracterelor din şir (lucrând indexat sau cu pointeri)
b) folosind funcţiile sistem (din bibliotecile string.h, stdlib.h, conio.h)

Utilizarea funcţiilor standard pentru lucrul cu şiruri de caractere

1.Determinarea lungimii unui şir de caractere

```
unsigned strlen(const char *sir);
```

ex:
```
char sir[]="informatica";
cout<<strlen(sir);   // va afişa valoarea 11
```

2.Copierea unui şir de caractere în alt şir de caractere

```
char* strcpy(char * destinatia, const char * sursa);
```

ex:
```
char sir[]="informatica",sir1[]="exemplu";
strcpy(sir, sir1);
cout<<sir;   // va afişa exemplu
```

Copierea unui prefix al unui şir de caractere în alt şir de caractere

```
char* strncpy(char * destinatia, const char * sursa,int n);
```

Va copia în destinaţie din sursă maxim n caractere; dacă şirul sursa are lungimea < n atunci şirul destinaţie va avea inclus şi caracterul NULL, altfel nu.

ex:
```
char sir[]="informatica",sir1[]="exemplu";
strncpy(sir, sir1, 7);
cout<<sir;
strncpy(sir1, sir ,12);
cout<<sir1;
```

3.Concatenarea a două şiruri de caractere

```
char* strcat(char * destinatia, const char * sursa);
// adaugă şirul sursă la sfârşitul şirului destinaţie, inclusiv NULL
char* strncat(char * destinatia, const char * sursa, int n);
// adaugă primele n carectere din şirul sursă la sfârşitul şirului destinaţie
```

ex:
```
char sir[]="om ",sir1[]="harnic ";
int n=3;
strcat(sir, sir1);
cout<<sir;
strncat(sir,sir1,n);
cout<<sir;
```

Obs.

1.Dimensiunea şirului destinaţie trebuie să fie suficient de mare pentru a reţine rezultatul.

2.Dacă dorim să concatenăm doar primele n caractere din şirul sursă la şirul destinaţie şi lungimea şirului sursă este <n atunci se va concatena întreg şirul sursă la destinaţie, inclusiv caracterul NULL.

4.Compararea a două şiruri de caractere

```
int strcmp(char * s1, char * s2);
// compară şirul s1 cu şirului s2 şi returnează, astfel:
```

- valoarea 0 dacă s1 este egal cu s2
- un întreg >0 dacă s1>s2 (dpdv lexicografic)
- un întreg <0 dacă s1<s2

```
int strncmp(char * s1, char * s2, int n);
// compară primele n caractere din cele două şiruri
```

ex:
```
char sir[]="harpa",sir1[]="harnic";
int n=3;
if (strcmp(sir, sir1)==0)
        cout<<"egale";
else if (strcmp(sir, sir1)<0)
        cout<<sir<<'<'<<sir1;
    else
        cout<<sir1<<'<'<<sir;
cout<<'\n';
if (strncmp(sir, sir1, n)==0)
        cout<<"sunt egale primele"<<n<<"caractere din sirurile"<<sir<<"si"<<sir1;
else if (strcmp(sir, sir1)<0)
        cout<<sir<<'<'<<sir1;
    else
        cout<<sir1<<'<'<<sir;
```

Obs.

Dacă dorim ca la comparare să nu se facă diferenţa între litere mari şi mici, atunci vom folosi funcţia stricmp.

```
int stricmp(char * s1, char * s2);
```

5.Căutarea primei apariţii a unui caracter într-un şir

```
char* strchr(char * s, char x);
```

//caută şi returnează un pointer către prima apariţie a caracterului x în şirul s; dacă caracterul x nu se găseşte în şirul s, funcţia va returna NULL

ex:
```
char sir[]="Sambata merg la CEX",x, *p;
x='a';
p=strchr(sir, x);
if (p!=0)
        cout<<x<<"apare prima data in "<<sir<<"pe pozitia"<<p-sir;
else
        cout<< x<<"nu apare deloc in "<<sir;
cout<<'\n';
```

6.Căutarea primei apariţii a unui şir în alt şir

```
char* strstr(const char * s1, const char * s2);
```

//caută şi returnează un pointer către prima apariţie a şirului s2 în şirul s1; dacă s2 nu se găseşte în şirul s1, funcţia va returna NULL
ex: char sir[]="Sambata merg la CEX",sir1[]="merg", *p;

```
p=strstr(sir, sir1);
if (p!=0)
        cout<<sir1<<"apare prima data in "<<sir<<"pe pozitia"<<p-sir;
else
        cout<< sir1<<"nu apare deloc in "<<sir;
cout<<'\n';
```

7. Transformarea literelor mici în litere mari şi invers

```
char* strlwr(char* s);
        // transformă toate literele şirului în litere mici
char* strupr(char* s);
        // transformă toate literele şirului în litere mari
```

8.Inversarea şirului

```
char* strrev(char* s);
        // inversează şirul în el însuşi
```

ex: char sir[]="sambata";

```
strrev(sir);
cout<<sir<<'\n';  // va scrie atabmas
```

9.Funcţii de conversie

Fişierul antet stdlib.h conţine funcţii care realizează conversia unui număr întreg sau real în şir de caractere şi invers.

```
char *  itoa(int n, char* sir, int baza);
char *  ltoa(long n, char* sir, int baza);
char *  utoa(unsigned long n, char* sir, int baza);
        // funcţiile permit conversia unui număr întreg în şir de caractere; numărul
           obţinut se va afla în baza indicată
int  atoi(char* sir);
long atol(char* sir);
double atof(char* sir);
long strtol(char* s, char* p, int baza);
        // funcţiile permit conversia în şir de caractere a unui număr întreg;
        // funcţia strtol returnează prin pointerul p adresa primului caracter din sirul s
        care nu a putut fi convertit
```

Probleme propuse

1.**Problema "ION"**
Vi se dau mai multe succesiuni cu acelaşi număr de litere mici pe care trebuie să le aşezaţi unele sub altele. În careul astfel format trebuie să căutaţi un anumit cuvânt. Căutarea se face atât pe orizontală (linie) cât şi pe verticală (coloană) iar cuvântul poate apărea de la dreapta la stânga, de la stânga la dreapta, de jos în sus sau de sus în jos. Să se numere liniile şi coloanele pe care cuvântul apare cel puţin o dată.

Cerinţă
Cunoscând cuvântul c care trebuie căutat, numărul n de succesiuni de n litere precum şi cele n succesiuni, se cere să se determine pe câte linii şi coloane apare cuvântul dat.

Date de intrare

Pe prima linie a fişierului de intrare cuvant.in se află un cuvânt c format din litere mici. Pe linia a 2-a se găseşte un număr natural n iar pe următoarele linii se dau cele n succesiuni de câte n litere mici fiecare.

Date de ieşire

Fişierul de ieşire cuvant.out va conţine numărul de linii şi coloane pe care apare cuvântul dat.

Restricţii
Cuvântul c are maxim 20 de litere iar n nu depaşeste 100.

Exemplu

<table>
<tr><th>cuvant.in</th><th>cuvant.out</th><th>Explicatii</th></tr>
<tr><td>ion
6
ionabc
ajonoi
uuiono
aixbnn
ionnoi
cnrsit</td><td>8</td><td>Aşezăm succesiunile de litere unele sub altele, ca în careul de mai jos:
<table><tr><td>i</td><td>o</td><td>n</td><td>a</td><td>b</td><td>c</td></tr><tr><td>a</td><td>j</td><td>o</td><td>n</td><td>o</td><td>i</td></tr><tr><td>u</td><td>u</td><td>i</td><td>o</td><td>n</td><td>o</td></tr><tr><td>a</td><td>i</td><td>x</td><td>b</td><td>n</td><td>n</td></tr><tr><td>i</td><td>o</td><td>n</td><td>n</td><td>o</td><td>i</td></tr><tr><td>c</td><td>n</td><td>r</td><td>s</td><td>i</td><td>t</td></tr></table>
Cuvântul ion se găseşte pe liniile 1 şi 3 (de la stanga la dreapta), pe linia 2 (de la dreapa la stânga), pe linia 5 (pe ambele direcţîî), pe coloanele 2 şi 6 (de sus în jos) şi pe coloanele 3 şi 5 (de jos în sus), deci în total pe 8 direcţii.</td></tr>
</table>

Timp de execuţie/test: 1 secundă

2. **Operaţii cu radicali**
Se citeşte o expresie aritmetică scrisă corect din punct de vedere matematic. Aceasta expresie conţine numai operatorii + şi -, iar operanzii pot avea doar una din formele :

n - număr natural fără semn
nrm - cu semnificaţia **n radical din m**, unde m si n sunt numere naturale fără semn, n>=2
rm - cu semnificaţia **radical din m**, unde m este număr natural fără semn.

Cerinţă
Se cere ca, pornind de la o expresie aritmetică de forma precizată anterior, să se efectueze calculele astfel încât să fie îndeplinite simultan condiţiile:

a) expresia are număr minim de radicali din numere libere de pătrate (expresia poate avea numai astfel de radicali) şi valoarea egală cu cea a expresiei iniţiale

b) numerele de sub radicalii expresiei de la punctual a) trebuie să apară în ordine crescătoare.

Date de intrare
Pe prima linie a fişierului de intrare radical.in se află o expresie aritmetică în forma precizată anterior.

Date de ieşire
Fişierul de ieşire cuvant.out va conţine expresia aritmetică adusă în forma cerută în enunţ.

Restricţii

- Expresia poate avea maxim 10000 de caractere şi este corectă din punct de vedere matematic.
- 0< n , m <1000
- Valorile care rămân sub radical, sunt numere naturale < 1000

Exemplu

radical.in	radical.out
4r12+r75-2r3+1+2r20+2r4	5+11r3+4r5

3.alfabetar
http://campion.edu.ro/arhiva/index.php?page=problem&action=view&id=1200
4.grad
http://campion.edu.ro/arhiva/index.php?page=problem&action=view&id=143
5.comp
http://campion.edu.ro/arhiva/index.php?page=problem&action=view&id=1241
6.ed
http://campion.edu.ro/arhiva/index.php?page=problem&action=view&id=296
7.secvsir
http://campion.edu.ro/arhiva/index.php?page=problem&action=view&id=969
8.cezar
http://campion.edu.ro/arhiva/index.php?page=problem&action=view&id=647