**Șiruri de caractere (stringuri)**

Un string se declară la fel ca un vector, elementele fiind de tip char.

Ex:

char s[14];

Elementele sunt indexate de la 0. Prima literă din string are indicele 0 and we cannot change this!!

O astfel de variabilă se poate accesa și la nivel de char, adică fiecare dintre valorile

s[0], s[1], ... este de tip char însă, are marele avantaj al accesării sale în întregime.

Astfel, dacă un vector de char este accesat prin numele variabilei (în cazul nostru s) de fapt accesăm adresa de început.

Toate contextele care permit această accesare se referă la caracterele din string care încep la adresa specificată și țin până la întâlnirea caracterului cu codul ASCII 0 = marca de final de string. A NU se confunda caracterul nul cu constanta NULL.

În orice constantă de tip string, acest caracter (marca 0) se subînțelege la închiderea ghilimelelor.

Ex:

"Th" înseamnă de fapt caracterele: 'T' (codul 84), 'h' (codul 104), '\0' (codul 0)

Astfel, trebuie să avem în vedere că orice string trebuie dimensionat cu numărul maxim de caractere care ne dorim să încapă în el PLUS 1.

Pe exemplul de mai sus "Th" are nevoie de 3 elemente.

**Citirea unui string:**

Fie stringul

char s[100];

1) Putem citi prin:

cin>>s; - citește în s până la primul caracter alb (spațiu, Enter, Tab). Este incapabilă să citească aceste caractere albe. Lucrurile necitite rămân pe buffer.

2)

cin.get(s,nr\_maxim\_caractere+1);

SAU

cin.getline(s,nr\_maxim\_caractere+1); (preferabil ăsta din urmă)

citește either nr\_maxim\_caractere either până dă de sfârșitul de linie. În principiu, la nivel bac, trebuie să-i dăm suficient de multe a.î. să citească până la sfârșitul de linie, de regulă punem aici tot atât cu cât îl declarăm.

!! În cazul unei citiri cu cin.get(...) sau cin.getline(...), dacă înainte de ea a fost făcută orice altă citire cu **cin>> ....** trebuie pus un cin.get(); ca să poată trece de sfârșitul de linie, în caz contrar ar citi un string vid.

Ex: sa presupunem că introducem:

7

Vlad behaie

int n;

char s[100];

cin**>>**n;

cin.getline(s,100); - ăsta NU se va citi corect, ci s va citi șirul vid.

Motivul: cin>>n; după ce l-a citit pe n NU a trecut la rând nou, ci a rămas fix după ultima cifră a lui n. cin.getline(s,100) a continuat din acel loc (care e sfârșitul liniei) până la sfârșitul liniei, deci nu citește nimic.

Corectarea se face printr-un cin.get();

**Obs**: forma (preferabil de utilizat in ORICE conditzii)

cin.getline(string,nr\_caractere+1);

acționează la fel ca cin.get(string,nr\_caractere+1) pentru citire însă, **DUPĂ** efectuarea citirii, trece automat la rândul următor (chiar dacă ar mai fi rămas caractere necitite pe linia curentă, le ignoră).

Astfel, dacă avem de citit mai multe propoziții (Care conțin și caractere albe) câte una de pe fiecare linie, e de preferat să citim cu cin.getline(...)

**Accesarea stringurilor prin intermediul pointerilor**

Dacă avem un string

char s[100];

prin s se înțelege adresa de început a stringului. (btw: s++ dă eroare deoarece s este o variabilă statică (așa este orice variabilă obișnuită) - adică are adresă fixă în memorie)

Una dintre cele mai de bază funcții care preiau adresa de început și prelucrează datele de‑acolo și până dau de marca de final de string este:

cout<<s;

Prin adunarea lui s cu un număr natural x se obțin adrese care sunt cu x poziții la dreapta lui s.

Ex:

char s[100]="Claudia";

cout<<s; //Claudia

cout<<s+1; //laudia

cout<<s+2; //audia

cout<<s+3; //udia

cout<<s+4; //dia

cout<<"Teleferic"+4; //afișează "feric"

**Pointerii** de tip char\* permit memorarea unor astfel de adrese de stringuri.

Practic, un astfel de pointer este ca o "căpușă" care "stă" pe o anumită adresă a stringului și prin intermediul căreia poate modifica (schimba) conținutul acestuia.

Pointerul de tip char \* se comportă EXACT ca un string însă spre deosebire de acesta POATE fi mutat.

Ex:

char s[100]="Marmota",\*p;

p=s+2; //obs: s=s+2 ar da eroare deoarece s este o variabilă string clasică

cout<<p; //rmota

p++;

cout<<p; //mota

p--;

cout<<p[0]; // 'r' obs: în cadrul unor expresii \*p <=> p[0]

cout<<p[1]; // 'm'

p[0]='l';

cout<<s;//Malmota

**Funcții cu stringuri**

**strlen(adresa);** - numărul de caractere din string

Ex:

char s[100]="Claudia",\*p;

cout<<strlen(s); //afișează 7

cout<<strlen(s+2); //afișează 5

p=s+2;

cout<<strlen(p+1); //afișează 4

**strcpy(adr1,adr2);** - copiază la adr1 caracterele care încep la adr2. După copiere pune și marca de sfârșit de string după caracterele copiate.

Ex:

char s[20]="teleferic pe cablu",q[20]="retrovizor";

strcpy(s+4,q+5);

cout<<s;// televizor

Iată o formă particulară care, pe sistemele de calcul actuale NU mai poate fi utilizată:

strcpy(s+i,s+j); !!(a NU se folosi la bac - în variantele din 2009 sunt o serie de exerciții care folosesc așa ceva, însă a fost declarată incorectă în versiuni mai noi de limbaj).

Această formă avea intenția de a șterge secvența de caractere care începe cu caracterul s[i] și se termină cu s[j-1].

Ex: (**DO NOT USE IT ANYMORE**)

char s[100]="Cotofana";

strcpy(s+2,s+5);

cout<<s; //afișează "Coana"

Obs: Forma particulară

strcpy(s+i,s+i+1);

era menită să șteargă DOAR caracterul de la indicele i.

!! **În ultima vreme NU se mai recomandă această funcție NICI pentru ștergere, deoarece implementarea sa pe arhitecturi paralele de calcul (cu mai multe procesoare) NU ia, neapărat, caracterele la rând** !!

atribuirea de mai sus se poate face cu ajutorul altui string:

char aux[..suficient de mare pt. s..];

strcpy(aux,s+i+1);

strcpy(s+i,aux);

La fel și exemplul anterior cu "Cotofana" ar putea fi scris astfel:

char s[100]="Cotofana",aux[100];

strcpy(aux,s+5);

strcpy(s+2,aux);

cout<<s; //afișează "Coana"

**strncpy(adr1,adr2,n);** - copiază la adr1 doar primele n caractere care încep la adr2 însă după copiere **NU** pune și marca de sfârșit de string după caracterele copiate(!)

char s[20]="farmacie",q[20]="doi cintezoi";

strncpy(s+2,q+6,4);

cout<<s;// fantezie -a se remarca faptul că, după caracterele copiate, "ntez",

NU a pus marca de final de string, ci restul stringului a rămas

pe când:

char s[20]="farmacie",q[20]="ntez";

strcpy(s+2,q);

cout<<s;// fantez -a se remarca faptul că, după caracterele copiate, "ntez",

a pus marca de final de string

**strcat(adr1,adr2);** - concatenează după caracterele stringului de la adr1 caracterele stringului de la adr2. După lipire pune și marca de sfârșit de string.

Ex:

char s[20]="iepure",p[20]="rechin";

strcat(s,p+2);

cout<<s;//afișează "iepurechin"

**strncat(adr1,adr2,n);** - concatenează după caracterele stringului de la adr1 primele n caractere ale stringului de la adr2. După lipire **pune** și marca de sfârșit de string.

Ex:

char s[20]="iepure",p[20]="chinuitor";

strncat(s,p,4);

cout<<s;//afișează "iepurechin"

• funcția **strchr**(string,char) caută prima apariție a lui char în string și întoarce ADRESA la care char-ul a fost găsit în string.

Dacă NU găsește întoarce NULL (!!NULL este o constantă specială și înseamnă adresă de memorie nulă - inexistentă)

Exemple:

cout<<strchr("Etienne",'e');//afișează enne

char s[30]="Steven",\*p;

p=strchr(s,'v');

cout<<p; //afișează ven

cout<<p-s; //afișează 3 (!!knd skdem adrese de memorie, se obține diferența de poziții dintre ele!!)

p[0]='f';p[1]='a'; ///în loc de p[0]='f' era fix același lucru: \*p='f'

cout<<p<<"\n"; //afișează fan

cout<<s<<"\n"; //afișează Stefan

Obs: funcția strchr poate fi foarte utilă atunci când dorim să verificăm dacă un caracter aparține unei anumite mulțimi de caractere, pentru că:

strchr(string\_cu\_mulțimea\_de\_caractere, caracter) are valoarea NULL dacă NU aparține, respectiv !=NULL dacă aparține.

Ex:

if(strchr("AEIOUaeiou",c)!=NULL) => e vocală

if(isalpha(c) && strchr("AEIOUaeiou",c)==NULL) => e consoană

• funcția **strstr**(string\_mare,string\_mic) caută prima apariție a lui string\_mic în string\_mare și întoarce ADRESA la care a fost găsit în.

Dacă NU găsește întoarce NULL (!!NULL este o constantă specială și înseamnă adresă de memorie nulă - inexistentă)

Exemple:

cout<<strstr("Apostolato Mara","to");//afișează tolato Mara

~~• funcția~~ **~~strupr~~**~~(string) transformă tot stringul, prin înlocuirea tuturor literelor mici cu litere mari. Obs: NU merge pe distribuțiile de C++ pe Linux shi nici pe pbinfo~~

~~Exemple:~~

~~char s[100]="Maya";~~

~~strupr(s);~~

~~cout<<s; "MAYA";~~

~~• funcția~~ **~~strlwr~~**~~(string) transformă tot stringul, prin înlocuirea tuturor literelor mari cu litere mari. Obs: NU merge pe distribuțiile de C++ pe Linux!~~

~~Exemple:~~

~~char s[100]="VlAd";~~

~~strupr(s);~~

~~cout<<s; "vlad";~~

~~• funcția~~ **~~strrev~~**~~(var\_string) inversează (oglindește) literele din var\_string.~~

~~Ex: Obs: NU merge pe distribuțiile de C++ pe Linux!~~

~~char s[10]="Vlad";~~

~~strrev(s);~~

~~cout<<s; //dalV~~

**Formarea unui string caracter cu caracter**

- ne luăm o variabilă care va avea la fiecare pas indicele la care urmează să punem un caracter. Fie ea nc. Inițial nc=0;

- de fiecare dată când vrem să adăugăm un caracter nou la string, atribuim:

v\_string[nc++]=caracterul\_nou;

- **la final, încheiem stringul**:

v\_string[nc]=0; '\0' <=> 0

**Funcția strtok**

Permite separarea unui string în "cuvinte" în funcție de un separator.

!!Operația distruge stringul de la care plecăm!!

**Modelul aplicării:**

- în urma aplicării lui strtok, prin intermediul unui pointer sunt întoarse adresele cuvintelor separate. Aceste adrese reprezintă de fapt locații din stringul dat (fiecare strtok va pune căpușa la un nou cuvânt)

- apelul inițial ("starter-ul" splituirii) se face prin:

strtok(s,"separatori")

Această funcție întoarce adresa primului cuvânt separat. Delimitarea se face până la primul dintre separatorii din grup care este întâlnit. Noua căutare va începe de după cuvântul separat, de la primul caracter al lui s care NU face parte din lista de separatori specificată.

- toate celelalte apeluri se fac prin:

strtok(NULL,"separatori")

Fiecare apel de acest fel va întoarce adresa următorului cuvânt separat. Când nu mai sunt cuvinte de separat, va întoarce NULL.

Ex:

char s[100]="Mara Stefania, APOSTOLATO",fn[100],mn[100],ln[100];

strcpy(fn,strtok(s,", "));

strcpy(mn,strtok(NULL,", "));

strcpy(ln,strtok(NULL,", "));

cout<<ln<<"\n"<<mn<<"\n"<<fn;

va afișa:

APOSTOLATO

Stefania

Mara

Dacă dorim separarea "automată", ne folosim de un pointer de tip char.

Scrierile pot fi de una dintre formele următoare:

**I.**

char \*p;

**p=strtok(s,"separatori");**

while(**p**)

{

//prelucram p

**p=strtok(NULL,"separatori");**

}

**II.**

char \*p;

for(**p=strtok(s,"sep.")**; **p** ; **p=strtok(NULL,"sep.")**)

{

//prelucram p

}

!!!NU PUTEM IMBRICA STRTOK !!

**Conversii**

Conversie = transformarea aceluiași conținut între diferite tipuri de date.

• atoi(string); - convertește stringul la număr int, atâta cât poate, adică cel mai mare

prefix al stringului care POATE fi interpretat ca număr întreg.

(atoi este în cstdlib)

Ex:

atoi("123"); → 123

atoi("123abc"); → 123

atoi("12.3"); → 12

atoi("abc12"); → 0

• atof(string); - convertește stringul la număr real, atâta cât poate, adică cel mai

mare prefix al stringului care POATE fi interpretat ca număr real.

(atof este în cstdlib)

Ex:

atof("123.4"); → 123.4

atof("12.3abc"); → 12.3

~~• itoa(int,var\_string,baza); - convertește numărul int, transformat în baza~~

~~specificată, în variabila string.~~

~~(itoa este în cstdlib - NICI ea nu merge pe Linux)~~

~~Ex:~~

~~itoa(100,s,10); - s primește "100"~~

~~itoa(106,s,16); - s primește "6a"~~

• sprintf(string,"%d",numar); - convertește numărul intreg în variabila string.

(sprintf este în cstdio)

• sprintf(string,"%f",numar); - convertește numărul real în variabila string.

Ex:

sprintf(s,"%f",3.14); - s primește "3.14"

**Vectori de stringuri**

Un vector de stringuri se declară ca o matrice de char-uri, adică:

char vs[nr\_stringuri][lmax];

Stringurile sunt indexate de la 0, însă putem lucra cu ele de la 1, la fel ca într-un vector de int. Lmax reprezintă lungimea maximă admisă pentru oricare dintre stringuri. Fiecare string va putea fi accesat ca orice string, prin vs[indice] și se comportă la fel.

Ex: Dacă vrem să facem un vector de stringuri în care să memorăm pe indicii de la 1 la 5 "Claudia", "Alberta", "Matei" și "Valyn", minimul declarației tre' să fie:

char nume[5][8];

strcpy(nume[1],"Claudia");//nume[1][0]='C';nume[1][1]='l';nume[1][2]='a';nume[1][3]='u';nume[1][4]='d';nume[1][7]=0;

strcpy(nume[2],"Alberta");

strcpy(nume[3],"Matei");

strcpy(nume[4],"Valyn");

cout<<strlen(nume[2]); //afiș. 7

cout<<nume[1][3]; //afiș. u

cout<<nume[4]+2; //afiș. "lyn"

**Pointeri către stringuri (căpușe:D)**

Dacă pointerul p are o adresă a unui string, am văzut deja că p[0] reprezintă primul caracter al acestui string. Acest p[0] este echivalent cu \*p.

De fapt, în C++, valabil nu doar pentru pointeri către char, ci pentru orice tip de pointeri (chiar și vectori obișnuiți)

x[i] = \*(x+i) = \*(i+x) = i[x]

Exemplu:

char s[10]="Andrei",\*p;

p=strchr(s,'n');

cout<<p<<"\n";//afish. "ndrei"

p++;

cout<<p<<"\n";//afish. "drei"

cout<<\*p<<"\n";//afish. d

(\*p)++; //asta este ca shi cum ash da p[0]++

cout<<\*p<<"\n";//afish. e

cout<<s<<"\n";//afish. Anerei

**Compararea lexicografică a două stringuri**

Se face ca în dicționar, însă, din păcate, spre deosebire de mai toate celelalte limbaje de programare, în C++ NU putem compara două stringuri cu <, <=, >, >=, == sau !=

Pentru compararea a două stringuri în C++ e obligatoriu să utilizăm funcția

strcmp(s1,s2)

Aceasta întoarce:

-1 dacă lexicografic, s1 este înainte de s2 (<)

0 dacă lexicografic, s1 este identic cu s2 (==)

1 dacă lexicografic, s1 este după s2 (>)

Ex:

strcmp("mama","bunica") → 1

strcmp("alb","albastru") → -1

strcmp("albastru","albi") → -1

strcmp("cici","cici") → 0

Obs: În C++ nici nu putem atribui unui string o valoare constantă cu operatorul = (în afară de declarația stringului, unde totuși se poate). Pentru asta folosim strcpy

|  |  |
| --- | --- |
| Așa nu:  **char s[100];**  **s="Horia";** | Așa da:  **char s[100];**  **strcpy(s,"Horia");** |

Obs: Funcția strlen(string) are de fapt EXACT următoarea implementare:

int strlen(char s[])

{

char \*p=s,l=0;

while(\*p)

{l++;p++;}

return l;

}

Asta înseamnă că, dacă parcurgem un string prin:

for(i=0;i<strlen(s);i++) **- DE EVITAT LA MAXIM această formă!!**

{....}

la fiecare repetare a for-ului se execută INTEGRAL întreaga funcție strlen, deci la fiecare pas de for se mai fac încă n pași.

Simpla folosire a lui strlen(s) pe post de condiție la for, va duce la timp n2 în loc de n pași.

Forme alternative:

**I.** n=strlen(s);

for(i=0;i<n;i++)

{...}

**II.**

for(i=0;s[i];i++)

{...}