Considerații teoretice

Formatori:
Tutor: Stângaciu Valentin
Tutor: Belu Claudiu-Marcel
+3

Data de începere a cursului:
25.09.2023
Utilizatori înscriși
Calendar
Note

Considerații teoretice

Un șir de caractere (string) este reprezentat în C prin vectori cu elemente de tip *char*. **Prin convenție, sfârșitul unui șir de caractere se consideră ca fiind caracterul cu codul 0 ('\0').** Funcțiile C folosesc acest caracter pentru a ști când s-a ajuns la sfârșit de șir. Se poate astfel folosi un vector mai mare (ex: de 100 caractere) pentru a se ține în el un șir de caractere de doar 9 litere.
Atenție: a nu se confunda terminatorul de șir ('\0', cod ASCII 0) cu cifra '0' (cod ASCII 48).

Folosind strict noțiunile de la vectori, putem inițializa un șir de caractere astfel:

```
char sir1[]={'s','a','l','u','t','\0'};
```

Limbajul C oferă o posibilitate mai simplă de a scrie șiruri de caractere, folosind **ghilimele**:

```
char sir1[]="salut";
```

Când se folosesc ghilimele, compilatorul adaugă automat terminatorul '\0' la sfârșitul șirului, deci acesta nu mai trebuie specificat. Deoarece vectorii sunt în marea majoritate a cazurilor identici cu pointerii, putem rescrie inițializarea ca fiind:

```
char *sir1="salut";
```

Constatăm astfel că în C, atunci când scriem caractere între ghilimele, tipul de date este "*char[]*" (chiar dacă avem un singur caracter, ex: "#"), iar acel șir are automat adăugat terminatorul de șir. Când scriem un caracter între apostroafe, tipul de date este "*char*" și avem de-a face întotdeauna cu un singur caracter, chiar dacă pentru a-l scrie este necesar să folosim secvențe escape (ex: '\\').

Afișarea șirurilor de caractere se poate face cu *printf*, folosind placeholderul %s, sau cu funcții gen *puts(s)*, care trece automat la linie nouă după ce a afișat șirul *s*.

Citirea unor șiruri de caractere se face în mai multe feluri, cu efecte diferite:

pentru a citi un singur cuvânt, până la primul caracter cu funcție de spațiu (spațiu, TAB, ENTER, etc), se folosește placeholderul %s în *scanf*. În acest fel putem de exemplu citi mai multe valori diferite, aflate pe aceeași linie.

pentru a citi o linie întreagă, până la apăsarea tastei ENTER, se folosește funcția *fgets(sir,nr_max_caractere,stdin)*, cu următoarele argumente:

sir - destinația unde vor fi citite caracterele (un vector de caractere)

nr_max_caractere - lungimea maximă a șirului care poate fi stocat în *sir*, incluzând terminatorul de șir. Întotdeauna vor fi citite maximum *nr_max_caractere-1*, pentru a rămâne loc pentru terminator, care este adăugat automat. După caracterele citite, *fgets* va pune automat în șir (dacă mai este loc) caracterul \n (newline). Astfel, în marea majoritate a cazurilor, dacă citim date cu *fgets*, trebuie să ținem cont de faptul că la sfârșitul șirului există un \n.

stdin - este o variabilă predefinită în C, care indică tastatura ca fiind sursa caracterelor citite (*standard input*). Când se va discuta despre fișiere, *stdin* va putea fi înlocuit cu un fișier, pentru a citi din acel fișier.

Observație: În C există și funcția *gets(sir)*, care într-un fel este o variantă simplificată a lui *fgets*. *gets* citește de la tastatură o linie, fără a ține cont de lungimea ei și o depune în *sir*. Această funcție nu este recomandat a fi folosită, deoarece ea nu testează dacă șirul citit încape în vectorul destinație, putând astfel apărea depășiri de vector, dacă se citesc prea multe caractere. Din acest motiv, compilatorul va emite o atenționare la

folosirea lui *gets*.

Funcția **scanf** poate și ea duce la depășire de vector când este folosită cu %s și este recomandat să se precizeze numărul maxim de caractere ce se pot citi: %20s

Exemplul 9.1: Să se scrie un program care citește o linie de text, îi convertește toate caracterele la litere mari și apoi afișează textul rezultat.

```
#include <ctype.h>
#include <stdio.h>
int main()
{
   char s[100];
   int i;
   printf("textul: ");
   fgets(s, 100, stdin);     // se citesc maxim 99 de caractere+\0; dacă sunt mai puține, la sfarsit se va depune și \n
   for (i = 0; s[i]; i++){      // se itereaza atata timp cat inca nu s-a ajuns la terminatorul de sir
       s[i] = toupper(s[i]);
   }
   printf("text final: %s", s);
   return 0;
}
```

Variabila *s* va conține șirul de caractere citit. Din definirea ei se constată că în ea vor putea încăpea maximum 99 de caractere și terminatorul. Iterarea într-un șir de caractere se face testând prezența terminatorului. Deoarece terminatorul are valoarea numerică 0, deci fals, când se ajunge la acesta, iterarea se oprește, fără ca terminatorul să fie inclus în procesare. În final *i* va fi poziționat chiar pe poziția terminatorului. Acest aspect este important dacă vrem să numărăm caracterele din șir, dacă vrem să adăugăm noi caractere, etc.

Aplicația 9.1: Să se citească o linie de la tastatură. Linia conține cuvinte care sunt formate doar din litere, cuvintele fiind despărțite prin orice alte caractere ce nu sunt litere. Să se capitalizeze prima literă din fiecare cuvânt și să se afișeze șirul rezultat.

Exemplul 9.2: Să se scrie un program care citește pe rând 2 nume. Cele două nume vor fi concatenate într-un alt șir de caractere cu „ și ” între ele și rezultatul va fi afișat.

```
#include <stdio.h>
int main()
{
   char s1[30], s2[30];                              // șirurile de intrare
   char s[100];                                          // șirul destinație
   char sep[] = " si ";   int i, j;
   printf("nume1: ");
   fgets(s1, 30, stdin);
   printf("nume2: ");
   fgets(s2, 30, stdin);
   j = 0;                                                        // index in destinatie
   for (i = 0; s1[i] && s1[i] != '\n'; i++){       // copiaza caracterele din s1 pana la aparitia \0 sau \n
       s[j++] = s1[i];
   }
   for (i = 0; sep[i]; i++){                           // concateneaza separatorul
       s[j++] = sep[i];
   }
   for (i = 0; s2[i] && s2[i] != '\n'; i++){       // concateneaza caracterele din s2
       s[j++] = s2[i];
   }
   s[j] = ‘\0’;                                                  // adauga terminatorul
   printf("text final: %s", s);
   return 0;
}
```

Deoarece operațiile cu șiruri de caractere sunt frecvente, în C există o bibliotecă de funcții pe șiruri de caractere, accesibilă prin intermediul antetului **<string.h>.** Aceste funcții în general încep cu „str” (de la **string**), urmat de un sufix care este prescurtarea unui verb ce definește acțiunea realizată. Printre aceste funcții enumerăm:

Funcție	**Efect**
strcpy(dst,src)	copiază șirul „src” peste „dst”, inclusiv terminatorul. Pentru a se ține minte care este sursa și care este destinația, se poate face comparația cu o atribuire, deci destinatia este în stânga: dst=src;
strcat(s1,s2)	concatenează „s2” la sfârșitul lui „s1” și adaugă terminatorul

strlen(s)	returnează dimensiunea caracterelor șirului „s”, fără terminator. ATENȚIE! Funcția returnează un unsigned long, ca urmare, NU COMPARAȚI NUMERE NEGATIVE cu strlen(s)!!
strcmp(s1,s2)	compară s1 cu s2 în ordine alfabetică, ținând cont de diferența între litere mari și mici. Dacă: s1 este înaintea lui s2 alfabetic, returnează o valoare negativă (<0) s1 este după s2 alfabetic, returnează o valoare strict pozitivă (>0) s1 este identic cu s2, returnează 0
stricmp(s1,s2)	la fel ca **strcmp**(s1,s2), dar nu ține cont de litere mari și mici
strchr(s,c)	Caută caracterul c în șirul s. Dacă l-a găsit, returnează un pointer la poziția sa, altfel returnează NULL.
strstr(sir,subsir)	Caută subșirul în șir. Dacă l-a găsit, returnează un pointer la poziția sa, altfel returnează NULL.
strcspn(s1,s2)	returnează câte caractere există de la începutul lui *s1* până la prima apariție în s1 a oricărui caracter din s2. De obicei s2 este o listă de separatori (spațiu, virgulă, ...) și atunci se returnează poziția primului separator găsit. Dacă în s1 nu a fost găsit niciun caracter din s2, se returnează dimensiunea lui s1.

Folosind aceste funcții, exemplul de mai sus poate fi rescris astfel:

```c
#include <string.h>
#include <stdio.h>
int main()
{
   char s1[30], s2[30];
   char s[100];
   char sep[] = " si ";
   printf("nume1: ");
   fgets(s1, 30, stdin);
   s1[strcspn(s1, "\n")] = '\0';                    // elimină posibilul \n, punand terminatorul de sir peste el
   printf("nume2: ");
   fgets(s2, 30, stdin);
   s2[strcspn(s2, "\n")] = '\0';
   strcpy(s, s1);                                     // seteaza sirul rezultat ca fiind primul nume
   strcat(s, sep);                                   // concateneaza separatorul
   strcat(s, s2);                                     // concateneaza al doilea nume
   printf("text final: %s", s);
   return 0;
}
```

Dacă se va consulta documentația C pentru aceste funcții, se va constata că ele sunt definite în următoarea manieră:

```c
int strcmp(const char *s1, const char *s2)
```

const înseamnă că acel argument este constant pentru funcție, adică funcția nu are dreptul să-l modifice.

În general implementarea acestor funcții este simplă și putem la rândul nostru să scriem funcții similare, dacă avem nevoie de ele. De exemplu, putem implementa *strcat(s1,s2)* în felul următor:

```c
// s1 va fi modificat prin concatenarea la el a caracterelor din s2
// s2 va fi doar citit, deci poate fi const
char *mystrcat(char *s1,const char *s2){
    char *p=s1;
    while(*p)p++;                                 // trece peste toate caracterele din s1
    // p este pozitionat pe terminatorul lui s1
    // itereaza toate caracterele din s2 si le concateneaza la s1
    for( ; *s2 ; p++, s2++)*p=*s2;
    // p este pozitionat pe prima pozitie de dupa caracterele concatenate
    *p='\0';                                        // adauga terminatorul
    return s1;                                     // strcat trebuie intotdeauna sa returneze s1
}
```

Dacă avem nevoie să păstrăm în memorie mai multe șiruri de caractere, putem fiecare șir să-l păstrăm în interiorul unei structuri care conține un câmp de tip șir de caractere. Pe lângă acest câmp, mai putem păstra în structură și alte informații, de exemplu contoare etc.

Exemplul 9.3: Să se scrie un program care citește linii de caractere până la întâlnirea liniei vide. Fiecare linie conține un cuvânt. Să se afișeze în ordine alfabetică cuvintele distincte introduse și de câte ori apare fiecare.

```
#include <stdio.h>
#include <string.h>
```

```
 typedef struct{
     char txt[20];   // textul cuvantului
     int cnt;            // nr de aparitii
     }Cuvant;
```

```
Cuvant cuvinte[100];    // vector de cuvinte
 int nCuvinte;       // nr de cuvinte din vector
```

```
// cauta un cuvant in vectorul de cuvinte
// daca il gaseste, returneaza adresa structurii sale
// daca nu, returneaza NULL
 Cuvant *cauta(char *txt)
{
     for(int i=0;i<nCuvinte;i++){
         if(strcmp(cuvinte[i].txt,txt)==0)return &cuvinte[i];
         }
     return NULL;
}
```

```
// sortare alfabetica cuvinte
 void sortare()
{
     char schimbare;
     do{
         schimbare=0;
         for(int i=1;i<nCuvinte;i++){
             if(strcmp(cuvinte[i-1].txt,cuvinte[i].txt)>0){
                 Cuvant c=cuvinte[i-1];
                 cuvinte[i-1]=cuvinte[i];
                 cuvinte[i]=c;
                 schimbare=1;
                 }
             }
         }while(schimbare);
}
```

```
int main()
{
     char txt[20];
     // citeste cate un cuvint pe linie, pana la sir vid
     for(;;){
         fgets(txt,20,stdin);
         txt[strcspn(txt, "\n")]='\0';
         if(strlen(txt)==0)break;
         Cuvant *c=cauta(txt);
         if(c){  // daca cuvantul este deja in vector, ii incrementeaza contorul
             c->cnt++;
             }else{  // daca cuvantul e nou, il adauga in vector
             strcpy(cuvinte[nCuvinte].txt,txt);
             cuvinte[nCuvinte].cnt=1;
             nCuvinte++;
             }
         }
     sortare();
     for(int i=0;i<nCuvinte;i++){    // afisare
         printf("%s: %d\n",cuvinte[i].txt,cuvinte[i].cnt);
         }
     return 0;
}
```

Când se citește cu *scanf* **un număr**, *scanf* sare peste caracterele de tip spațiu (spațiu, ENTER, TAB, etc) de dinaintea numărului. Toate cifrele din număr sunt citite, până la caracterul ENTER de după număr. Acesta, nefiind o cifră, nu mai este adăugat la număr, ci rămâne în așteptare (în bufferul de intrare, fără a fi consumat). Dacă după numărul citit se citește un alt număr este în regulă, deoarece noul *scanf* va sări la rândul său peste caracterele de tip spațiu existente, deci și peste ENTER-ul rămas în așteptare. Dacă însă după ce s-a citit un număr se va citi un șir de caractere (ex: cu *fgets*), ENTER-ul rămas în așteptare va termina automat, încă de la început (înainte de a se citi orice caracter) șirul ce se dorește a fi citit, și deci va rezulta un șir vid. Pentru a se evita aceasta, după fiecare citire de număr cu *scanf* se poate folosi *getchar* pentru a se consuma ENTER-ul rămas după citirea numărului respectiv.

◄ Teme și aplicații

Sari la...

Teme și aplicații ►