

PROGRAMAREA CALCULATOARELOR TIPUL DE DATE CHARACTER ȘI ȘIRURI DE CHARACTERE ÎN LIMBAJUL C

Prelegere

Kulev Mihail, dr., conf. univ.

Stimați studenți și stimată audiență!

Mă numesc Kulev Mihail sunt doctor, conferințiar universitar la Universitatea Tehnică a Moldovei. Din cadrul cursului PROGRAMAREA CALCULATOARELOR Vă propun spre atenția Dumneavoastră prelegerea cu tema:

” TIPUL DE DATE CHARACTER ȘI ȘIRURI DE CHARACTERE ÎN
LIMBAJUL C

PROGRAMAREA CALCULATOARELOR

TIPUL DE DATE CHARACTER ȘI ȘIRURI DE CHARACTERE ÎN LIMBAJUL C

Prelegere

Kulev Mihail, dr., conf. univ.



TIPUL DE DATE CHARACTER ȘI ȘIRURI DE CARACTERE ÎN LIMBAUL C

Conținutul prelegerii

1. Tipul de date caracter.

Vom afla ce reprezintă tipul de date caracter și standardul de codificare a caracterilor ASCII, modalități de declarare și de inițializare a caracterilor și ce reprezintă caracterele speciale tipărite și netipărite.

2. Tablouri și șiruri de caractere.

Vom determina prin ce se diferă tablouri (vectori) de caractere și șiruri de caractere (stringuri) și vom demonstra exemple și modalități de declarare și inițializare a șirurilor de caractere și a pointerilor spre tipul char

3. Funcții standard de prelucrare a caracterilor și a șirurilor de caractere.

Vom studia și analiza prototipurile funcțiilor standard de prelucrare a caracterelor și a șirurilor de caractere și vom demonstra exemple de utilizare a funcțiilor.

Exemple de cod.

Pe parcursul studierii întrebărilor menționate vom prezenta exemple de cod în limbajul C.

1. Tipul de date caracter

Valorile asociate tipului de date caracter (**char**) sunt elemente din mulțimea caracterelor – litere, cifre, caractere de punctuație și altele, care sunt utilizate, în special, pentru scrierea și citirea textelor, conform unei codificări numerice a setului disponibil de caractere. Codificarea este dependentă de compilator și de sistemul de operare folosite.

Declararea unei variabile de tip caracter (**char**) în limbajul C are forma: **char nume;**

- unde **char** este denumirea tipului, iar **nume** reprezintă denumirea variabilei caracter. Tipul **char** poate fi compatibil fie cu tipul **signed char**, fie cu tipul **unsigned char**.

O variabilă caracter ocupă în memorie un singur octet (8 biți) sau un byte, care conține codul caracterului. Acest cod este o valoare numerică și depinde de sistemul de codificare a caracterelor.

Cel mai utilizat sistem de codificare a caracterilor este codul ASCII (American Standard Code for Information Interchange), care se utilizează în limbajul C.

Codul ASCII (American Standard Code for Information Interchange)

- cod standard de reprezentare a caracterelor ca valori întregi pe 7 sau 8 biți ai unui octet.
- introdus pentru a **se obține** o compatibilitate între tipuri diferite de echipamente folosite la procesarea datelor.
- Codul ASCII standard (pentru tipul char sau tipul signed char)
 - constă din 128 de numere întregi (reprezentate pe 7 biți ai unui octet, cu valori între 0 și 127) atribuite caracterelor alfabetului latin, cifrelor, semnelor de punctuație, celor mai uzuale caractere speciale și ale unor coduri de control (comenzi) netipăribile (primele 32 de caractere).
- Codul ASCII extins (pentru tipul char sau tipul unsigned char)
 - constă de asemenea din 128 numere întregi, cu valori între 128 și 255 (pentru reprezentarea lor folosind toți cei 8 biți ai unui octet), care reprezintă caractere suplimentare din alte limbi, simboluri matematice, grafice, caractere speciale sau simboluri ale unor monede străine.
- Folosirea acestor coduri face posibilă prelucrarea caracterelor în limbajul C.

Declarare și inițializare a variabilelor de tip caracter

În programe scrise în limbajul C caracterele pot fi tratate ca întregi și invers. Fiecare constantă caracter (sau literal caracter) este asociat cu o valoare întreagă, codul caracterului în setul de caractere ASCII. Un literal caracter se reprezintă prin simbolul caracterului respectiv inclus între apostrofi (dacă este caracter tipăribil). Exemple de declarare și inițializare a variabilelor de tip caracter: **char litera = 97; // declarare și inițializare cu codul numeric ASCII**

char litera = 'a'; // declarare și inițializare cu constanta caracter (literal de tip char).

char litera; litera = 'B'; // declarare și

// atribuire în timpul rulării programului

Caracterele speciale și de control (de comandă)

se reprezintă prin mai multe caractere numite **secvențe escape** (se încep cu simbolul \).

'\n' - sfârșit de linie (LF), rând nou (NL); **'\t'** - tabulare orizontală (HT);

'\v' - tabulare verticală (VT); **'\b'** - revenire la caracterul precedent (BS);

'\r' - revenire la început de linie (CR); **'\f'** - pagină nouă la imprimantă (FF); **'\a'** - alarmă (BEL);

'\\' - caracterul \ ; **'\?'** - caracterul ? ; **'\''** - caracterul ' ; **'\"'** caracterul " ;

'\ooo' - caracterul cu codul octal ooo ;

'\xhh' caracterul cu codul hexazecimal hh.

Diferite reprezentări ale caracterelor în limbajul C

Exemple:

- Simbol grafic: `'A'` `'\x41'` `'\101'` (sunt echivalente)
- Caracterele (`\` ' " ?): (`'\\'` `'\x5c'` `'\134'`) (`'\''` `'\x27'` `'\47'`) (`'\"'` `'\x22'` `'\42'`)
(`'\?'` `'\x3f'` `'\77'`) (sunt echivalente)
- Caracter de comandă: `'\n'` `'\xa'` `'\12'` (sunt echivalente)
- Nul octet (octet zero): `'\0'` `'\x00'` (sunt echivalente)

2. Tablouri și șiruri de caractere

În limbajul C nu există un tip de date șir de caractere (string), deși există constante șir de caractere – orice șir de caractere scris între ghilimele. **Definiție:** Un șir de caractere este un vector de caractere terminat cu caracterul '\0' numit nul octet (octet zero).

Există însă, anumite particularități în lucrul cu șiruri de caractere față de lucrul cu alte tablouri unidimensionale (vectori).

Prin natura lor, șirurile pot avea o lungime variabilă în limite foarte largi, iar lungimea lor se poate modifica chiar în cursul execuției unui program, ca urmare a unor operații cum ar fi concatenarea a două șiruri, ștergerea sau inserția într-un șir .

Pentru simplificarea listei de parametri și utilizării șirurilor în funcții s-a decis că fiecare șir memorat într-un vector (tablou 1-D) să fie terminat cu un caracter numit terminator de șir, caracterul '\0' ce are codul ASCII egal cu zero și astfel să nu se mai transmită explicit lungimea șirului în funcție. Multe funcții care produc un nou șir precum și funcțiile standard de citire adaugă automat un octet terminator la șirul produs (citit), iar funcțiile care prelucrează

sau afișează șiruri detectează sfârșitul șirului la primul octet zero.

Exemplu: Șirul de caractere "Anul 2020" ocupă 10 octeți, ultimul fiind '\0'.

Șiruri de caractere (stringuri)

Am constata că în limbajul C nu există variabile de tip string, dar este posibil să se proceseze șiruri de caractere. Un șir în limbajul C reprezintă un set de caractere stocate într-un tablou 1-D (alocat static sau dinamic) de tip caracter și terminat cu un octet nul (octet zero) `'\0'` și, un astfel de set de caractere, este considerat un singur obiect - un șir. Dacă octetul nul nu este anexat la sfârșitul unui set de caractere stocate în tablou, setul respectiv de caractere nu este considerat un șir (string). Numele tabloului alocat static și numele pointerului pentru tabloul alocat dinamic sunt de asemenea utilizate ca numele de șiruri stocate în aceste tablouri. De menționat, că numele unui șir este o constantă pointer (pentru un tablou alocat static) sau variabilă pointer (pentru tabloul dinamic), ambele reprezintă adresa primului caracter din șir (adresa începutului șirului în memorie). De aceea, fiecare caracter al șirului poate fi accesat folosind acest nume și operator de indexare `[]` sau operator de inderectare `*`. În limbajul C un string literal (șir constant) se scrie între ghilimele. Rețineți că șirurile constante sunt alocate în memoria fixă (zona de constante) și nu pot fi modificate.

Șiruri de caractere (stringuri)

Exemplu de declarare a unui tablou de caractere și de inițializare a tabloului cu șirul constant: **char c[]="cuvant";**
Aici observăm folosirea ghilimelelor pentru șirul constant. Această instrucțiune va aloca static memoria de 7 octeți pentru tabloul de tip caracter, în care se va reprezenta (se va copia) șirul constant de caractere "cuvant" alocat în memoria fixă de 7 octeți (6 octeți pentru 6 litere șirului constant și 1 octet pentru caracterul '\0').

Dacă dorim să alocăm un spațiu de memorie mai mare (pentru a putea folosi tabloul în scopul de a stoca șiruri de caractere mai lungi), putem folosi o declarație de tipul **char c[10] = "cuvant";** Astfel, am alocat spațiu suficient pentru un șir de 9 caractere. O inițializare în formă **char c[6] = "cuvant";** în care spațiul alocat este egal cu numărul de caractere, nu va determina compilatorul să genereze un warning/o eroare, acest lucru poate avea rezultate neașteptate, dacă în memorie – după ultimul element al tabloului nu se află (întâmplător) valoarea 0 (null octet).

Exemplu de declarare unui pointer spre tipul **char** și de inițializare a pointerului cu adresa unui șir constant:

```
char *pc="cuvant";
```

În acest exemplu memoria este alocată static pentru variabila pointer **pc** de tip **char *** și acest pointer este inițializat la compilare cu adresa șirului constant **"cuvant"** alocat în memoria fixă.

Exemplu de declarație unui pointer spre tipul **char** și de atribuire a pointerului adresei unui șir constant:

```
char *pc;    pc="cuvant";
```

În acest exemplu memoria este alocată static pentru variabila pointer **pc** de tip **char*** fără inițializare la compilare și ulterior (la execuție) pointerului i se atribuie adresa șirului constant **"cuvant"**.

Pointerul **pc** poate fi inițializat (poate avea) și cu adresa unui spațiu (tablou) de tip char alocat dinamic.

De menționat că în limbajul C nu există operația de atribuire de șiruri de caractere (sau în general de tablouri), ci numai atribuire de pointeri.

Exemplu de declarație unui tablou de caractere și de inițializare tabloului în mod caracter cu caracter la compilare: **char c[]= {'c','u','v','a','n','t','\0'};**

Concluzie: există două posibilități de definire a șirurilor de caractere:

- ca tablou de caractere;

Exemple: char sir1[30]; char sir2[10]="exemplu"; char sir3[]= {'e','x','e','m','p','l','u','\0'};

- ca pointer la caractere (inițializat cu adresa unui șir sau a unui spațiu alocat dinamic). La definire se poate face și inițializare:

Exemple:

char *sir4; // sir4 trebuie inițializat cu adresa unui șir sau a unui spațiu de tip char alocat dinamic (pe heap)

sir4=sir1; // sir4 ia adresa unui șir static

sir1=sir4 ; // greșit!

sir4=sir2; sau sir4=&sir2; sau sir4=&sir2[0]; // sunt echivalente

sir4=(char *)malloc(100); // se alocă dinamic un spațiu de tip char pe heap

char *sir5="test"; // sir5 este inițializat cu adresa șirului constant

3. Funcțiile standard de prelucrare a caracterelor și a șirurilor de caractere

Funcțiile standard de intrare/ieșire pentru caractere (din stdio.h):

- **scanf("%c",...); printf("%c",...);** utilizează specificatorul de format %c

- **getchar (); putchar ();**

int getchar(void);

- returnează codul unui caracter citit de la tastatura sau valoarea EOF (constantă simbolică definită în stdio.h, având valoarea -1) dacă s-a tastat Ctrl/Z.

int putchar(int c);

- tipărește pe ecran caracterul transmis ca parametru;
- returnează codul caracterului sau EOF în cazul unei erori.

Funcții de intrare/ieșire pentru șiruri de caractere (din stdio.h)

`scanf("%s", s);`

- citește caractere până la întâlnirea primului blanc sau Enter; acestea nu se adaugă la șirul s;
- plasează '\0' la sfârșitul lui s;
- dacă se tastează CTRL/Z returnează EOF;
- codul lui blanc sau Enter rămân în buffer-ul de intrare

`printf("%s", s);`

- tipărește șirul s

Funcții de intrare/ieșire pentru șiruri de caractere (din stdio.h)

char * gets(char * s);

- citește caractere până la întâlnirea caracterului Enter; acesta nu se adaugă la șirul s;
- plasează '\0' la sfârșitul lui s;
- returnează adresa primului caracter din șir;
- dacă se tastează CTRL/Z returnează NULL;
- codul lui Enter e scos din buffer-ul de intrare

int puts(char * s);

- tipărește șirul s, trece apoi la rând nou
- întoarce o valoare nenegativă, sau EOF la insucces

Observații

Nu se recomandă citirea caracter cu caracter a unui șir (cu specificatorul de format “%c” cu funcția `scanf(“%c”,..)` sau cu funcția `getchar()`) decât după apelul funcției “`fflush(stdin)`” care golește zona tampon de citire.

În caz contrar se citește caracterul „\n” (cod 10), care rămâne în zona tampon după citire cu `scanf(“%s”,..)` sau cu `getchar()`.

Pentru a preveni erorile de depășire a zonei alocate pentru citirea unui șir se poate specifica o lungime maximă a șirului citit în funcția `scanf()`.

Exemplu:

```
char nume[30]; while (scanf (“%29s”, nume) != EOF) printf (“%s \n”, nume);
```

numai primele 29 de caractere vor fi citite, pentru ca al 30-lea caracter va fi „\0”.

Funcții standard de prelucrare a caracterelor (din ctype.h)

Pentru a ușura lucrul cu șiruri de caractere, în biblioteca standard C sunt prevăzute o serie de funcții, ale căror prototipuri sunt date în fișierele <ctype.h> și <string.h>.

În fișierul <ctype.h> există o serie de funcții care primesc un parametru de tip caracter și întorc rezultatul 1 sau egal cu 0, după cum caracterul argument satisface sau nu condiția specificată:

islower(c) 1 dacă c din {'a'..'z'}; **isupper(c)** 1 dacă c din {'A'..'Z'} ;

isalpha(c) 1 dacă c din {'A'..'Z'} sau din{'a'..'z'}; **isdigit(c)** 1 dacă c din {'0'..'9'};

isxdigit(c) 1 dacă c din {'0'..'9'}sau{'A'..'F'} sau {a'..'f'};

isalnum(c) 1 dacă isalpha(c) || isdigit(c); **isspace(c)** 1 dacă c din {' ', '\n', '\t', '\r', '\f', '\v'};

isgraph(c) 1 dacă c este afișabil, fără spațiu; **isprint(c)** 1 dacă c este afișabil, cu spațiu;

isctrl(c) 1 dacă c este caracter de control;

ispunct(c) 1 dacă isgraph(c) && !isalnum(c).

Conversia din literă mare în literă mică și invers se face folosind funcțiile:

tolower(c) și **toupper(c)**.

Exemplu: Scrieți o funcție care convertește un șir de caractere reprezentând un număr întreg, într-o valoare întreagă. Numărul poate avea semn și poate fi precedat de spații albe.

```
#include <ctype.h>

int atoi(char *s)
{ int i, nr, semn;
  for(i=0; isspace(s[i]); i++);    // ignora spatii albe
  semn=(s[i]==-1)?-1:1;           // stabilire semn
  if(s[i]=='+' || s[i]=='-')       // se sare semnul
    i++;
  for(nr=0; isdigit(s[i]); i++)    // conversie in cifra
    nr=10*nr+(s[i]-'0');          // si alipire la numar
  return semn*nr;
}
```

Funcții standard de prelucrare a șirurilor de caractere (din string.h)

```
int strcmp(const char *s1, const char *s2);  
int stricmp(const char *s1, const char *s2);  
char *strcpy(char *d, const char *s);  
char* strncpy(char *d, const char *s, unsigned n);  
char *strdup(const char *s);  
int strlen(const char *s);  
char *strcat(char *d, const char *s);  
char *strncat(char *d, const char *s, unsigned n);  
char *strchr(const char *s, int c);  
char *strrchr(const char *s, int c);  
char *strstr(const char *s, const char *subsir);  
char* strtok(char *s1, const char *s2);  
int atoi(char* s);  
double atof(char* s);
```



int strcmp(const char *s1, const char *s2); returnează: <0, dacă s1 < s2; 0, dacă s1 = s2 ; >0, dacă s1 > s2;

int strncmp (const char *s1,const char *s2, int n); comparare a două șiruri pe lungimea n;

char *strcpy(char *d, const char *s); copiază șirul sursa s în șirul destinație d; returnează adresa șirului destinație;

char* strncpy(char *d,const char *s,int n); copiază maxim n caractere de la sursă la destinație; returnează adresa șirului destinație;

int strlen(const char *s); returnează lungimea șirului fără a număra caracterul terminator

char* strcat(const char *d, const char *s); concatenează cele doua șiruri și returnează adresa șirului rezultat

char* strchr(const char *s,char c); returnează poziția primei apariții a caracterului c în șirul s, respectiv NULL dacă c nu este în s

char* strstr(const char *s, const char *ss); returnează poziția primei apariții a șirului ss în șirul s, respectiv NULL dacă ss nu e în s.

Funcțiile standard strcpy() și strcat() adaugă automat terminatorul zero la sfârșitul șirului produs de funcție!

Funcțiile pentru operații pe șiruri nu pot verifica depășirea memoriei alocate pentru șiruri,

deoarece primesc numai adresele șirurilor; cade în sarcina programatorului să asigure memoria

necesară rezultatului unor operații cu șiruri.

char *strdup(const char *s); Alocă memorie la o altă adresă și copiază în această memorie șirul s și întoarce adresa noului șir. Exemplu:

```
char * strdup ( char * adr) { int len=strlen(adr); // lungime șir de la adresa adr  
char * rez = (char*) malloc((len+1)*sizeof(char)); // alocă memorie pentru șir și terminator  
strcpy (rez,adr); // copiaza șir de la adr la adresa rez  
return rez; // rezultatul este adresa duplicatului }  
  
const char*
```

Argumentele ce reprezintă adrese de șiruri care nu sunt modificate de funcție. Interpretat ca “pointer la un șir constant (nemodificabil)”. Cuvântul cheie **const** în fața unei declarații de pointer cere compilatorului să verifice că funcția care are un astfel de argument nu modifică datele de la acea adresă.

Observație

Pentru realizarea unor noi operații cu șiruri se vor folosi pe cât posibil funcțiile existente:

// sterge n caractere de la adresa “d”

```
char * strdel ( char *d, int n)
```

```
{ if ( n < strlen(d)) { char *aux=strdup(d+n); strcpy(d,aux); }
```

```
return d; }
```

// inserează șirul s la adresa d

```
void strins (char *d, char *s)
```

```
{ char aux[20]; strcpy(aux,d); // creare copie șir care începe la adresa d
```

```
strcpy(d,s); // adauga șirul s la adresa d
```

```
strcat(d,aux); // concatenează la noul șir vechiul șir }
```

Funcția strtok()

Definiție cuvânt sau atom lexical (“token”): un șir de caractere separat de alte șiruri prin unul sau câteva caractere cu rol de separator între cuvinte (de exemplu, spații albe);

char *strtok(char *str1, const char *str2);

- pentru extragere de cuvinte ce pot fi separate și prin diferite caractere (‘,’ sau ‘;’).
- are ca rezultat un pointer la următorul cuvânt din linie și adaugă un octet zero la sfârșitul acestui cuvânt, dar nu mută la altă adresă cuvintele din text.
- acest pointer este o variabilă locală statică în funcția strtok(), deci o variabilă care își păstrează valoarea între apeluri succesive.
- trebuie folosită cu atenție deoarece modifică șirul primit și nu permite analiza în paralel a două sau mai multe șiruri.

Exemplu: utilizarea funcției strtok()

```
int main ( )
{ char linie[128], *cuv;                // adresa cuvant în linie char
*sep=". , ; \t \n "                    // șir de caractere separator
gets (linie);                          // citire linie
cuv = strtok (linie, sep);              // primul cuvant din linie
while (cuv !=NULL)
{ puts (cuv);                          // scrie cuvant
cuv = strtok(0, sep); // urmatorul cuvant din linie
} return 0; }
```

Erori posibile la utilizarea șirurilor

- Utilizarea unei variabile pointer neinițializate în funcția “scanf” (sau “gets”), datorită confuziei dintre vectori și pointeri.
- Poate cea mai frecventă eroare de programare
- Nu se manifestă întotdeauna ca eroare la execuție
- Exemplu greșit: `char * s;` // corect este: `char s[80];` 80= lung. maximă
`scanf (“%s”,s);` // citește la adresa conținută în “s”

Erori posibile la utilizarea șirurilor

Compararea adreselor a două șiruri în locul comparației celor două șiruri

- eroare frecventă (nedetectată la compilare)

Exemplu: `char a[50], b[50]; // aici se memoreaza doua siruri`

`scanf ("%49s%49s", a,b); // citire siruri a si b`

`if (a==b) printf("egale\n"); //gresit,rezultat zero`

- Pentru comparare corectă de șiruri se va folosi funcția `strcmp()`.

Exemplu : `if (strcmp(a,b)==0) printf ("egale\n");`

Aceeași eroare se poate face și la compararea cu un șir constant.

Exemple: `if (nume == ".") break; ...} // gresit !`

`if (strcmp(nume, ".") == 0) break;... } // corect`

Erori posibile la utilizarea șirurilor

Atribuirea între pointeri cu intenția de copiere a unui șir la o altă adresă

O parte din aceste erori pot fi semnalate la compilare.

Exemple:

```
char a[100], b[100], *c ; // memorie alocata dinamic la adresa "c"
```

```
c = (char*) malloc(100);
```

```
a = b; // eroare la compilare
```

```
c = a; // corect sintactic dar nu copiaza sir (modifica "c")
```

```
strcpy (c,a); // copiaza sir de la adresa "a" la adresa "c"
```

```
strcpy (a,b); // copiaza la adresa "a" sirul de la adresa "b"
```

Probleme propuse spre rezolvare pentru prelucrarea caracterelor și șirurilor de caractere

1. Scrieți variante de implementare a funcțiilor de bibliotecă `strlen`, `strcmp`, `strcpy` și `strcat`.
2. Să se scrie un program care:
 - citește cuvinte tastate fiecare pe câte un rând nou, până la CTRL/Z (varianta: pana la introducerea unui cuvânt vid)
 - afișează cuvântul cel mai lung
 - afișează cuvintele ce încep cu o vocală
3. Se citesc trei șiruri `s1`, `s2` și `s3`. Să se afișeze șirul obținut prin înlocuirea în `s1` a tuturor aparițiilor lui `s2` prin `s3`. (Observație: dacă `s3` este șirul vid, din `s1` se vor șterge toate subșirurile `s2`).
4. Scrieți o funcție care convertește un întreg într-un șir de caractere în baza 10.
5. Scrieți un program care elimină caracterul dat dintr-un șir de caractere.
6. Scrieți un program care determină dacă cuvântul dat este polindrom.

Tutoriale online pentru tema prelegerii:

1. <https://ocw.cs.pub.ro/courses/programare/laboratoare/lab10>
2. https://profs.info.uaic.ro/~infogim/2017/lectii/6/65_caractere.pdf
3. <https://igotopia.ro/cum-programezi-cu-siruri-de-caractere-in-limbajul-c/>
4. <http://docshare04.docshare.tips/files/20868/208687039.pdf>
5. <https://sites.google.com/site/razvanaciu/programarea-calculatoarelor/programarea-calculatoarelor---10---siruri-de-caractere>
6. <https://www.pbinfo.ro/articole/19/siruri-de-caractere-in-c>
7. <http://andrei.clubcisco.ro/cursuri/1pc/curs/1/Curs%202a.pdf>
8. <http://info.tm.edu.ro:8080/~junea/cls%2010/siruri%20de%20caractere/teorie%20siruri%20de%20caractere.pdf>
9. https://info64.ro/Siruri_de_caractere/



FACULTATEA
CALCULATOARE, INFORMATICĂ
ȘI MICROELECTRONICĂ

FCIM

VĂ MULȚUMESC PENTRU ATENȚIE!

MULTĂ SĂNĂTATE ȘI SUCCESE!