# 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

În limbajul C caracterele individuale sunt reprezentate folosind tipul de date **char**. Fiecărui caracter i se atribuie un cod numeric (**ASCII** – American Standard Code for Information Interchange), cu ajutorul căruia se pot codifica 128 de caractere. În tabela de mai jos pentru fiecare caracter se dau codul său zecimal și cel hexazecimal (în baza de numerație 16). Se poate constata că unele caractere nu sunt reprezentabile direct ci ele sunt „caractere de control" gen „spațiu", „linie nouă", „TAB", „ENTER", etc:

| Decimal | Hex | Char | Decimal | Hex | Char | Decimal | Hex | Char | Decimal | Hex | Char |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | [NULL] | 32 | 20 | [SPACE] | 64 | 40 | @ | 96 | 60 | ` |
| 1 | 1 | [START OF HEADING] | 33 | 21 | ! | 65 | 41 | A | 97 | 61 | a |
| 2 | 2 | [START OF TEXT] | 34 | 22 | " | 66 | 42 | B | 98 | 62 | b |
| 3 | 3 | [END OF TEXT] | 35 | 23 | # | 67 | 43 | C | 99 | 63 | c |
| 4 | 4 | [END OF TRANSMISSION] | 36 | 24 | $ | 68 | 44 | D | 100 | 64 | d |
| 5 | 5 | [ENQUIRY] | 37 | 25 | % | 69 | 45 | E | 101 | 65 | e |
| 6 | 6 | [ACKNOWLEDGE] | 38 | 26 | & | 70 | 46 | F | 102 | 66 | f |
| 7 | 7 | [BELL] | 39 | 27 | ' | 71 | 47 | G | 103 | 67 | g |
| 8 | 8 | [BACKSPACE] | 40 | 28 | ( | 72 | 48 | H | 104 | 68 | h |
| 9 | 9 | [HORIZONTAL TAB] | 41 | 29 | ) | 73 | 49 | I | 105 | 69 | i |
| 10 | A | [LINE FEED] | 42 | 2A | * | 74 | 4A | J | 106 | 6A | j |
| 11 | B | [VERTICAL TAB] | 43 | 2B | + | 75 | 4B | K | 107 | 6B | k |
| 12 | C | [FORM FEED] | 44 | 2C | , | 76 | 4C | L | 108 | 6C | l |
| 13 | D | [CARRIAGE RETURN] | 45 | 2D | - | 77 | 4D | M | 109 | 6D | m |
| 14 | E | [SHIFT OUT] | 46 | 2E | . | 78 | 4E | N | 110 | 6E | n |
| 15 | F | [SHIFT IN] | 47 | 2F | / | 79 | 4F | O | 111 | 6F | o |
| 16 | 10 | [DATA LINK ESCAPE] | 48 | 30 | 0 | 80 | 50 | P | 112 | 70 | p |
| 17 | 11 | [DEVICE CONTROL 1] | 49 | 31 | 1 | 81 | 51 | Q | 113 | 71 | q |
| 18 | 12 | [DEVICE CONTROL 2] | 50 | 32 | 2 | 82 | 52 | R | 114 | 72 | r |
| 19 | 13 | [DEVICE CONTROL 3] | 51 | 33 | 3 | 83 | 53 | S | 115 | 73 | s |
| 20 | 14 | [DEVICE CONTROL 4] | 52 | 34 | 4 | 84 | 54 | T | 116 | 74 | t |
| 21 | 15 | [NEGATIVE ACKNOWLEDGE] | 53 | 35 | 5 | 85 | 55 | U | 117 | 75 | u |
| 22 | 16 | [SYNCHRONOUS IDLE] | 54 | 36 | 6 | 86 | 56 | V | 118 | 76 | v |
| 23 | 17 | [ENG OF TRANS. BLOCK] | 55 | 37 | 7 | 87 | 57 | W | 119 | 77 | w |
| 24 | 18 | [CANCEL] | 56 | 38 | 8 | 88 | 58 | X | 120 | 78 | x |
| 25 | 19 | [END OF MEDIUM] | 57 | 39 | 9 | 89 | 59 | Y | 121 | 79 | y |
| 26 | 1A | [SUBSTITUTE] | 58 | 3A | : | 90 | 5A | Z | 122 | 7A | z |
| 27 | 1B | [ESCAPE] | 59 | 3B | ; | 91 | 5B | [ | 123 | 7B | { |
| 28 | 1C | [FILE SEPARATOR] | 60 | 3C | < | 92 | 5C | \ | 124 | 7C | \| |
| 29 | 1D | [GROUP SEPARATOR] | 61 | 3D | = | 93 | 5D | ] | 125 | 7D | } |
| 30 | 1E | [RECORD SEPARATOR] | 62 | 3E | > | 94 | 5E | ^ | 126 | 7E | ~ |
| 31 | 1F | [UNIT SEPARATOR] | 63 | 3F | ? | 95 | 5F | _ | 127 | 7F | [DEL] |

Tipul de date **char** este tratat ca un număr întreg și astfel se pot realiza cu el orice operații numerice posibile numerelor întregi. Afișarea/citirea unui singur caracter se face cu placeholder-ul **%c** în *printf/scanf*. La *printf*, dacă se folosește %d se va afișa codul ASCII al caracterului, iar dacă se folosește %c se va afișa reprezentarea sa grafică (ex: o literă). Caracterele (tip de date *char*) se scriu între apostroafe:

```
char c='A';     // atribuie valoarea 'A' (codul ASCII 65) variabilei c
```

Se poate citi un caracter și cu funcția „**getchar**()", care returnează un caracter citit de la tastatură, sau se poate scrie cu „**putchar**(c)".

**Atenție:** Între **apostroafe** se scrie doar **un singur caracter** și tipul rezultat este **char.** Între **ghilimele** se scrie un **șir de caractere** și tipul rezultat este **char[]** (vector de caractere).

**Exemplul 10.1:** Să se citească un caracter de la tastatură și să se afișeze dacă este cifră, literă mică, literă mare sau altceva. În caz că e litera mică să se afișeze litera mare corespunzătoare și invers.

```c
#include <stdio.h>
int main()
{
   char c;
   printf("introduceti un caracter: ");
   scanf("%c", &c);
   if (c >= '0' && c <= '9')
       printf("este o cifra");
   else if (c >= 'a' && c <= 'z')
       printf("este o litera mica -> %c", c - 'a' + 'A');
   else if (c >= 'A' && c <= 'Z')
       printf("este o litera mare -> %c", c - 'A' + 'a');
   else
       printf("este altceva");
   return 0;
}
```

Pentru a se testa apartenența la o anumită clasă de caractere s-au folosit operații clasice de testare a apartenenței la un interval, la fel ca pentru numere. Pentru transformarea din numere mici în numere mari și invers s-au folosit tot operații numerice: prima oară s-a scăzut din *c* începutul intervalului din care face parte, pentru a se obține astfel deplasamentul (offset-ul) lui *c* față de începutul de interval, iar apoi s-a adăugat la acest offset începutul noului interval în care vrem să-l translatăm.

Deoarece astfel de operații pe caractere (clasificare, transformare) sunt destul de frecvente, limbajul C oferă o bibliotecă de funcții pe caractere accesibilă prin intermediul antetului **<ctype.h>** (*char types*), printre care enumerăm:

| **Funcție** | **Efect** |
|---|---|
| **toupper**(c)/**tolower**(c) | returnează litera mare/mică, corespunzătoare literei *c* |
| **isdigit**(c) | **true** dacă *c* este o cifră ('0'..'9') |
| **isalpha**(c) | **true** dacă *c* este o literă, mică sau mare ('a'..'z' sau 'A'..'Z') |
| **islower**(c) | **true** dacă *c* este o literă mică ('a'..'z') |
| **isupper**(c) | **true** dacă *c* este o literă mare ('A'..'Z') |

Folosind aceste funcții, exemplul anterior se poate scrie:

```c
#include <ctype.h>
#include <stdio.h>
int main()
{
   char c;
   printf("introduceti un caracter: ");
   scanf("%c", &c);
   if (isdigit(c))
       printf("este o cifra");   else if (islower(c))
       printf("este o litera mica -> %c", toupper(c));
   else if (isupper(c))
       printf("este o litera mare -> %c", tolower(c));
   else
       printf("este altceva");
   return 0;
}
```

**Observație:** Se poate constata că tabela ASCII, fiind inițial gândită de americani, nu conține coduri pentru caractere naționale, altele decât cele englezești (ex: românești, germane, chirilice, chinezești, grecești, etc). Pentru reprezentarea acestor caractere se folosește o altă codificare numită **UNICODE**, care atribuie un cod fiecărui caracter existent în lume, inclusiv caracterelor matematice, muzicale, etc. În limbajul C aceste coduri, care nu încap în tipul **char** (1 octet) se reprezintă folosind tipul **wchar_t** („**wide char type**", 2-4 octeți). Funcțiile pentru ele sunt în antetul **<wctype.h>** și au forma **iswalpha**, **iswupper**, **towupper**, etc. Pentru tipărire/citire există funcțiile **wprintf**/**wscanf**. În acest laborator ne vom ocupa doar de variantele ASCII ale acestor funcții, codificate cu caractere de tip **char**.

Anumite caractere din tabela ASCII, caractere ce nu sunt printabile, pot fi referite prin anumite secvențe, așa-numitele *secvențe escape*, care constă din caracterul **** (backslash), urmat imediat (fără spațiu) de un caracter care va interpretat în mod special, conform următorului tabel:

| **Secvență escape** | **Valoare ASCII** | **Efect** |
|---|---|---|
| **\n** | 0x0A in Linux<br>0x0A si 0x0D in Win | Mută cursorul la începutul liniei noi (newline) |

| | | |
|---|---|---|
| **\"** | 0x22 | Afișează ghilimele |
| **** | 0x5C | Afișează **** (backslash) |
| **\t** | 0x09 | Mută cursorul cu un TAB la dreapta |

## Operațiuni de elementare de citire/scriere caractere

Fiecare program are acces implicit la 3 "fișiere" pentru operațiuni de intare ieșire (scriere/citire de la tastatura/ecran):

- **stdin** – standard input – intrarea standard a programului – toate funcțiile standard de intrare vor citi implicit de la intrarea standard (se poate spune aproximativ că vor citi, generic, de la tastatură)
- **stdout** – standard output – ieșirea standard a programului – toate funcțiile standard de ieșire vor scrie implicit la ieșirea standard (se poate spune aproximativ ca vor scrie pe ecran, ca vor tipări)
- **stderr** – standard error – ieșirea standard de eroare a programului – similar cu stdout dar se folosește pentru a "tipări" erori

Momentan, putem considera faptul că *stdout* reprezintă ecranul sau terminalul iar *stdin* reprezintă tastatura. Practic putem spune cu ușurință că prin "printăm la stdout (standard output, ieșirea standard)" de fapt scriem pe ecran sau scriem în terminal. De asemenea, putem spune că prin "citim de la stdin (standard input, intrarea standard") de fapt citim de la tastatură.

Aceste fișiere speciale (stdin, stdout, stderr) sunt deschise și disponibile pentru fiecare program pe tot parcursul execuției programului într-un mod transparent pentru programator. Acestea sunt gestionate de sistemul de operare iar programatorul practic nu trebuie să facă nimic în privința lor.

Este important de menționat că atât intrarea standard cât și ieșirea standard nu sunt "conectate" direct la programele ce rulează ci între acestea sistemul de operare intervine cu serie de memorii tampon, buffere.
Așadar, atunci când un program utilizator scrie la ieșirea standard, datele ajung de fapt într-un buffer intern al sistemului de operare și acesta din urmă decide când anume acele date vor fi efectiv scrise la standard output.
La fel se întâmplă și în cazul intrării standard și anume atunci când scriem la tastatură datele ajung într-un buffer intern al sistemului de operare înainte să ajungă la programul nostru și doar sistemul de operare decide când va transfera datele către programul nostru.

Decizia sistemului de operare de trimiterea datelor din bufferele intermediare efectiv către stdout sau stdin se întâmplă prin golirea acestor buffere iar acest procedeul se cheamă flush (adica golirea bufferelor). Momentul în care apare operațiunea de flush a fișierelor stdin și stdout este decis de sistemul de operare. Decizia este de obicei luată de sistemul de operare în momentul în care buffer-ul se umple sau când primește anumite cereri din partea programului.

Operațiunea de flush a bufferelor de la stdin și stdout este determinată de următorii factori:

- umplerea bufferelor
- caracterul \n
- caracterul EOF

Acest lucru poate implica anumite comportamente. Spre exemplu, daca scriem un mesaj cu funcția printf acesta nu va fi scris în exact momentul apelului funcției printf ci ori când se umple bufferul de la stdout ori când se trimite către stdout caracterul \n. Ca și o consecință, dacă dorim ca mesajul nostru să ajungă la stdout în exact acel moment este necesar ca la sfârșitul mesajului să fortăm o linie nouă punând și caracterul \n.

Același lucru se întâmplă și în cazul intrării standard stdin. Dacă scriem ceva la tastatură, programul nostru va primi datelor ori în cazul în care se umple bufferul sistemului de operare, ori când trimitem caracterul \n prin tasta ENTER ori când trimitem caracterul EOF prin combinația tastelor CTRL+D (în Linux).

Este bine ca aceste aspecte să fie luate în calcul atunci când se implementează și se testează programele.

O importantă observație este faptul că fișierul stderr (ieșirea standard de eroare) este conectată direct la programele utilizatorului fără buffere intermediare.

Biblioteca standard C pune la dispoziție 2 funcții pentru citire/scriere elementară de caractere.

Funcția putchar(int c) primește ca argument caracterul pe care îl va scrie la stdout, în cazul nostru, în terminal.

Funcția getchar(int c) are rolul de a citi un caracter de la intrarea standard (stdin), în cazul nostru de la tastatură. Această funcție returnează caracterul citit ca și un întreg cu semn (int) și valoarea EOF (End Of File) în caz de eroare sau în cazul în care fișierul stdin nu mai este disponibil. Funcția aceasta este cu blocare în sensul că la un apel ea nu returnează până ce nu citește un caracter de la stdin sau până găsește caracterul EOF. Caracterul EOF poate fi obținut de la tastatură prin combinația tastelor CTRL+D pentru sisteme Linux și CTRL+Z pentru sisteme Windows.

Considerăm următorul exemplu, în care se implementează un program ce citește caracter cu caracter de la intrarea standard (stdin) și îl afișează la ieșirea standard:

```
#include <stdio.h>
int main(void)
{
    int c = 0;
    while ((c = getchar()) != EOF)
    {
        putchar(c);
    }
    return 0;
}
```

Testarea programului se poate face astfel: după rulare programul așteaptă introducerea de caractere. Se pot introduce caractere dar acestea nu sunt „trimise" programului decât ori după un anumit număr de caractere ori după introducerea caracterului \n. Așadar, pentru testare, se pot introduce caractere iar la următoarea apăsare a tastei ENTER bucla while va fi executată și caracterele introduse vor fi scrise la stdout. Programul se va termina doar la recepționarea caracterului EOF. Acesta poate fi provocat din tastatură prin CTRL+D în sisteme Linux.

## Operațiuni de redirectare

Sistemul de operare Linux oferă o serie de mecanisme prin care utilizatorul poate foarte ușor să înlocuiască ieșirea standard sau intrarea standard cu fișiere fără ca programul să "știe" acest aspect. Acest lucru se poate realiza prin semnele "<" si ">" folosite în terminal.

Redirectarea ieșirii standard într-un fișier. Exemplu:

```
valy@staff:~$ ./program > file.txt
```

În această situație, absolut orice este scris de către programul *program* la ieșirea standard (stdout) prin apelul oricărei funcții precum printf sau putchar va ajunge de fapt în fișierul file.txt . Dacă fișierul nu există sistemul de operare în va crea iar dacă există atunci conținutul acestuia va fi sters.

Redirectarea intrării standard dintr-un fișier existent

```
valy@staff:~$ ./program < file.txt
```

În această situație, către ieșirea standard a programului *program* va fi redirectat conținutul fișierului file.txt. Astfel, orice operațiune de citire de la intrarea standard (stdin) prin apelul funcțiilor precum scanf sau getchar() va citi practic din acest fișier. Când programul va citi tot fișierul, acesta va primit caracterul EOF. Este absolut necesar ca fișierul ce este redirectat la intrarea standard a unui program să existe, în caz contrar, sistemul de operare va genera un mesaj de eroare în terminal și nu va lansa în execuție programul

Este important de menționat că programul "nu stie" faptul că i s-au redirectat fișierele standard de intrare și de ieșire. Deci programul "nu știe" că în urma unei scrieri la stdout el defapt scrie într-un fișier.

Acest lucru poate fi folosit ca un mare avantaj deoarece se poate ușura considerabil procedura de testare a programelor ce lucrează cu standard output și standard input.

Putem aplica aceste considerente în testarea programului dezvoltat anterior. În locul testării clasice de la tastatură putem redirecta către stdin un fișier de test realizat în prealabil. Testarea deci se poate realiza extrem de util astfel:

```
valy@staff:~$ gcc -Wall -o program in_out_2.c
```

```
valy@staff:~$ ./program < testfile.txt
```

◄ Test grilă 2

Sari la...

Teme și aplicații ►

Contactați serviciul de asistență

Sunteți conectat în calita
S1-L-AC-CTIRO1-PC

Meniul meu
Profil
Preferinte
Calendar
ZOOM