DEBRECENI EGYETEM
INFORMATIKAI KAR

Formázott kimenet/bemenet használata C nyelven

Dr. Varga Imre

Kimenet

A formázott output legfontosabb eszközei C nyelven a `printf()`, az `fprintf()` és az `sprintf()` függvények. A visszatérési érték a sikeresen kiírt karakterek száma, illetve hiba esetén egy negatív érték. Egy karakter egy bájtot jelent, de vannak több karakteres szimbólumok is (pl. escape karakterek). Ezek a függvények változó paraméterszámúak. Mindegyiknek van egy ún. formátumsztring paramétere, amely meghatározza, hogy nézzen ki a kimeneti adatfolyamba kerülő karaktersorozat. Ennek a formátumsztringnek meghatározott helyeire különböző típusú értékek szúrhatók be megfelelő alakban. Ilyenkor az adott típusú értéket a függvény mindig karaktersorozattá alakítja. Az így kiegészített (formátumsztringből és további értékekből meghatározott) karaktersorozat kerül a kimeneti adatfolyamba. A `printf()` esetén a képernyőre (`stdout`), `fprintf()` esetén egy írásra megnyitott szöveges fájlba, `sprintf()` esetén egy sztringbe. A függvények definíciói (amelyek az `stdio.h` header állományban vannak megadva) a következők:

```
int printf (const char * formatstring, ...);
int fprintf (FILE* stream, const char * formatstring, ...);
int sprintf (char* str, const char * formatstring, ...);
```

A továbbiakban egy példagyűjteményt találsz, amely összefoglalja a formátumsztringben használt speciális kifejezéseket.

Konstans sztring kiírása a „kurzor” aktuális pozíciójától kezdve. (Ebben az egyszerű esetben alternatívaként használható a `puts()` függvény, de ez automatikusan egy sortörést is helyez a szöveg végére, azaz mindig külön sort ír ki, szemben a `printf()` függvénnyel).

```
printf("Hello World!");
```

← C kódrészlet

```
Hello World!
```

← kimenet

Speciális karakterek használata

`\n`: Sortörés (New Line, Line Feed, LF). Kiírás során új sor megkezdése. A `'\n'` is egy jelenlévő karakter (egyetlen bájt), de mivel „nem nyomtatható”, így látszólag hiányzik.

```
printf("Hello\n World!");
```

```
Hello
World!
```

`\r`: Kocsi vissza (Carriage Return, CR). A kurzor aktuális pozíciójának az adott sor elejére állítása, így a további írás a sor elején valósul meg. Bizonyos esetekben a szövegfájlokban a soremelés egy CR+LF karakterkombinációval van megvalósítva.

```
printf("abcdef\rABC");
```

```
A B C d e f
```

\t: Tabulátor. Vízszintes kihagyásként jelenik meg. Mérete változó, adott pozíciótól egy megadott következő pozícióig tart. Egyetlen karakter, nem azonos egy szóközsorozattal.

```
printf("A\tZ\nabc\txyz");
```

```
A       Z
a b c   x y z
```

\a: Bell vagy Alert. Hatására egy alapértelmezett figyelmeztető hang szólal meg. Nem nyomtatható karakter, tehát vizuálisan nem jelenik meg.

```
printf("\a");
```

”Ding”

\b: Visszatörlés (Backspace). A sorban a kurzor aktuális pozíciója előtt álló karakter kitörli, ezzel a kurzort eggyel visszább mozdítja.

```
printf("abcdx\rdef");
```

```
a b c d e f
```

\": Macskaköröm karakter. Egy sztring kezdetét és végét macskaköröm karakter jelzi. Ha magát a macskaköröm karaktert akarjuk kiírni, akkor ez az escape karakter használandó.

```
printf("The word \"read\" is a verb.");
```

```
T h e   w o r d   " r e a d "   i s   a   v e r b .
```

\\: Backslash karakter. A visszaperjel speciális jelentéssel bír, egy escape karakter kezdetét jelzi. Ha magát a visszaperjel karaktert akarjuk kiírni, akkor ez a speciális escape karakter használandó.

```
printf("Windows path: C:\\Users\\John\\x.txt");
```

```
W i n d o w s   p a t h :   C : \ U s e r s \ J o h n \ x . t x t
```

%%: Százalékjel karakter. A formátumsztringben a százalékjel karakternek speciális jelentése van, ezért, ha magát ezt a karaktert akarod kiírni, akkor azt duplán kell írni.

```
printf("15%%duck 85%%chicken");
```

```
15 % d u c k   8 5 % c h i c k e n
```

Fixpontos számábrázolású értékek behelyettesítése

%d: Előjeles egész szám beszúrása tízes számrendszerben megadva. A %i is használható.

```
int NumMoons=95;
printf("Jupiter has %d moons.",NumMoons);
```

```
J u p i t e r   h a s   9 5   m o o n s .
```

`%6d`: Előjeles egész szám beszúrása legalább 6 karaktert felhasználva szükség esetén vezető szóközökkel. A megjelenítési hossz módosítható. Például jobbra igazított hasáboknál hasznos, így helyiérték helyes lesz a felírás.

```
int A=12, B=345, C=6, D=7890;
printf("%6d\n%6d\n%6d\n%6d\n",A,B,C,D);
```

```
    12
   345
     6
  7890
```

`%02d`: Előjeles egész szám beszúrása legalább 2 karaktert felhasználva szükség esetén vezető nullával. A megjelenítési hossz módosítható. Fontos, ha adott darabszámú számjegy kell (például számként tárolt bankszámlaszám, telefonszám).

```
int h=8, m=3, s=0;
printf("%02d:%02d:%02d",h,m,s);
```

```
08:03:00
```

`%-4d`: Előjeles egész szám beszúrása legalább 4 karaktert felhasználva (szükség esetén utána szóközökkel). A megjelenítési hossz módosítható. Például balra igazított hasáboknál hasznos.

```
int a=21, b=9, c=136;
printf("%-4dapples\n%-4dbananas\n%-4dcherries",a,b,c);
```

```
21  apples
9   bananas
136 cherries
```

`%+d`: Előjeles egész szám beszúrása úgy, hogy a pozitív előjel is mindig explicit módon ki legyen írva. (Ha az előjel negatív, az mindig megjelenik.)

```
int N=210;
printf("%+d vs %d",N,N);
```

```
+210 vs 210
```

`%u`: Egy egész szám beszúrása mindenképpen előjelnélküli értelmezéssel.

```
int N=-1;
printf("%u vs %d",N,N);
```

```
4294967296 vs -1
```

`%x`: Egy egész szám beszúrása tizenhatos számrendszerben megadva. Az `a-f` számjegyek kisbetűvel megjelenítve.

```
int A=700, B=0x700;
printf("%x != %x",A,B);
```

```
2bc != 700
```

`%X`: Egy egész szám beszúrása tizenhatos számrendszerben megadva. Az `A-F` számjegyek nagybetűvel megjelenítve.

```
int A=700, B=0x700;
printf("%X != %X",A,B);
```

```
2BC != 700
```

`%#x`: Egy egész szám beszúrása tizenhatos számrendszerben megadva és `"0x"` előtaggal kiegészítve.

```
int A=700, B=0x700;
printf("%#x != %#x",A,B);
```

```
0x2BC != 0x700
```

`%o`: Egy egész szám beszúrása nyolcas számrendszerben megadva.

```
int A=436, B=0664;
printf("%o == %o",A,B);
```

```
664 == 664
```

`%#o`: Egy egész szám beszúrása nyolcas számrendszerben megadva és a szokásos `"0"` előtaggal kiegészítve.

```
int A=436, B=0664;
printf("%#o == %#o",A,B);
```

```
0664 == 0664
```

`%ld`: Egy 8 bájtos előjeles egész szám (`long int`) beszúrása tízes számrendszerben megadva. (A `long long int` típus esetén a `%lld` megadása szükséges.)

```
long int BigNumber=1000000000000;
printf("%ld != %d",BigNumber,BigNumber);
```

```
1000000000000 != -727379968
```

`%hi`: Egy 2 bájtos egész szám beszúrása tízes számrendszerben megadva.

```
short int A=32767;
printf("%hi and %hi",A,A+1);
```

```
32767 and -32768
```

`%c`: Egy karakter beszúrása. A karakter megadható egybájtos egész számként (karakter kód) is.

```
char c='A';
printf("%c and %c",c,66);
```

```
A and B
```

`%p`: Egy memóriacím beszúrása prefixszel ellátott tizenhatos számrendszerbeli értékként. (A mutatók értékei is fixpontos ábrázolású adatok.)

```
int i=123;
int *P=&i;
printf("address of i: %p",P);
```

```
address of i: 0x7fffe6add9b8
```

Lebegőpontos számábrázolású értékek behelyettesítése

`%f`: Egyszeres pontosságú lebegőpontos érték (`float`) beszúrása 6 tört tizedesjeggyel megjelenítve, az utolsó helyiértéken kerekítést alkalmazva. (Figyelembe kell venni a lebegőpontos számábrázolás pontatlanságát. Például a `9876.54321` float érték `9876.542969` számként jelenik meg.)

```
float x=-12.3, y=0.000000567;
printf("%f %f",x,y);
```

```
-12.300000 0.000001
```

`%11f`: Egyszeres pontosságú lebegőpontos érték (`float`) beszúrása legalább 11 karakteren, ezen belül alapértelmezetten 6 tört tizedesjeggyel megjelenítve, az utolsó helyiértéken kerekítést alkalmazva. A megjelenítési hossz módosítható. Például jobbra igazított hasáboknál hasznos, így helyiérték helyes lesz a felírás.

```
float x=-12.34, y=0.000000567;
printf("%11f\n%11f",x,y);
```

```
  -12.340000
    0.000001
```

`%11.3f`: Egyszeres pontosságú lebegőpontos érték (`float`) beszúrása legalább 11 karakteren, ezen belül 3 tört tizedesjeggyel megjelenítve, az utolsó helyiértéken kerekítést alkalmazva. A megjelenítési hossz módosítható. Például jobbra igazított hasáboknál hasznos, így helyiérték helyes lesz a felírás.

```
float x=-12.345678, y=0.0004321;
printf("%11.3f\n%11.3f",x,y);
```

```
    -12.346
      0.000
```

`%-11.3f`: Egyszeres pontosságú lebegőpontos érték (`float`) beszúrása legalább 11 karakteren, ezen belül 3 tört tizedesjeggyel megjelenítve, az utolsó helyiértéken kerekítést alkalmazva és balra igazítva. A megjelenítési hossz módosítható.

```
float x=123.45;
printf("%11.3fUSD",x);
```

```
123.450   USD
```

`%011f`: Egyszeres pontosságú lebegőpontos érték (`float`) beszúrása legalább 11 karakteren jobbra igazítva, szükség esetén vezető nullákkal kiegészítve, ezen belül alapértelmezetten 6 tört tizedesjeggyel megjelenítve, az utolsó helyiértéken kerekítést alkalmazva. A megjelenítési hossz módosítható.

```
float x=12.34, y=0.12345678;
printf("%11f\n%11f",x,y);
```

```
  0  0  1  2  .  3  4  0  0  0
  0  0  0  0  .  1  2  3  4  5  7
```

`%lf`: Dupla pontosságú lebegőpontos érték (`double`) beszúrása 6 tört tizedesjeggyel megjelenítve, az utolsó helyiértéken kerekítést alkalmazva.

```
float x=123.45678987654;
printf("%f %f",x,y);
```

```
1  2  3  .  4  5  6  7  9  0
```

`%e`: Lebegőpontos érték beszúrása tudományos alakban `'e'` jelöléssel. Ez egyfajta normálalak: `1.234560e+03` → $1.23456\times10^{+3}$.

```
float x=1234.56,0.000987;
printf("%e %e",x,y);
```

```
1.234560e+03  9.869999e+04
```

`%E`: Lebegőpontos érték beszúrása tudományos alakban `'E'` jelöléssel. Ez egyfajta normálalak: `9.869999E-04` → 9.87×10^{-4}.

```
float x=1234.56,0.000987;
printf("%E %E",x,y);
```

```
1.234560E+03  9.869999E+04
```

`%g`: Lebegőpontos érték beszúrása úgy, hogy az értékhez képest minimális számú tizedesjeggyel legyen megjelenítve (azaz lehagyja a törtrészben szereplő záró nullákat). Ha a törtrész 0, akkor még a tizedespontot sem írj ki. (Ha kedvezőbb, a tudományos alakot használ. Ekkor `%g` esetén `'e'` jelölést alkalmaz `%G` esetén `'E'` jelölést.)

```
float w=123.0;
float x=123.450000;
float y=12.3456789;
float z=12345678.9;
printf("%g\n%g\n%g\n%g",w,x,y,z);
```

```
1  2  3
1  2  3  .  4  5
1  2  .  3  4  5  7
1  .  2  3  4  5  7  e  +  0  7
```

Sztringek behelyettesítése

`%s`: Egy karaktersorozat behelyettesítése a formátumsztringbe. Annyi karaktert szúr be, amilyen hosszú az adott sztring.

```
char name[10]="John";
printf("Hello %s!",name);
```

```
H  e  l  l  o     J  o  h  n  !
```

`%8s`: Egy karaktersorozat behelyettesítése a formátumsztringbe legalább 8 karakter hosszan, jobbra igazítva, balról szóközöket beszúrva. A megjelenítési hossz módosítható.

```
char name[10]="John";
printf("Hello %s!",name);
```

```
H e l l o       J o h n !
```

`%-8s`: Egy karaktersorozat behelyettesítése a formátumsztringbe legalább 8 karakter hosszan, balra igazítva, jobbról szóközöket beszúrva. A megjelenítési hossz módosítható.

```
char name[10]="John";
printf("Hello %s!",name);
```

```
H e l l o   J o h n       !
```

Színek és stílusok kezelése

`\033[0m`: Ez az ANSI escape code az alapértelmezett színt és normál stílust állítja vissza a konzolban. (Eddig tart a korábban megváltoztatott szín és stílus hatása.)

```
printf("\033[0mColored text!");
```

```
C o l o r e d   t e x t !
```

`\033[1m`: A `'['` után szereplő számjegy a szöveg stílusát határozza meg. Például az `1` a félkövér stílust jelenti. Amíg más érték nem kerül beállításra, addig minden további `printf()` hívás a legutóbbi beállítást fogja alkalmazni. A stílust leíró számjegyek jelentését a következő táblázat foglalja össze.

stílus számjegy	leírás
0	normál (reset)
1	félkövér
2	sötétített
3	dőlt
4	aláhúzott
5, 6	villogó
7	invertált (előtér és háttér szín csere)
8	rejtett
9	áthúzott

```
printf("\033[1mColored text!");
```

```
C o l o r e d   t e x t !
```

`\033[;31m`: A `';'` után szereplő szám a szöveg (előtér) színét határozza meg. Például az `31` a piros színt jelenti. Amíg más érték nem kerül beállításra, addig minden további `printf()` hívás a legutóbbi beállítást fogja alkalmazni. Az előtér színét leíró számok jelentését a következő táblázat foglalja össze.

színkód	előtér (betű) szín
30	fekete
31	piros
32	zöld

33	sárga
34	kék
35	magenta
36	cián
37	fehér

```
printf("\033[;31mColored text!");
```

Colored text!

`\033[;;42m`: A második `';'` után szereplő szám a háttér színét határozza meg. Például az `42` a zöld színt jelenti. Amíg más érték nem kerül beállításra, addig minden további `printf()` hívás a legutóbbi beállítást fogja alkalmazni. A háttér színét leíró számok jelentését a következő táblázat foglalja össze.

színkód	háttér szín
40	fekete
41	piros
42	zöld
43	sárga
44	kék
45	magenta
46	cián
47	fehér

```
printf("\033[;;42mColored text!");
```

Colored text!

`\033[3;34;43m`: Dőlt, kék színű szöveg, sárga háttérrel. A szöveg megjelenítése után minden alapállapotba kerül. (A `"\033["` után és az `'m'` előtt három szám adható meg: stílus, előtérszín, háttérszín. Ezek közé pontosvesszőt kell tenni. Ha valamelyik érték hiányzik, akkor legutóbbi beállítás marad érvényben.)

```
printf("\033[4;34;43mColored text!\033[0m");
```

Colored text!

Bemenet

A formázott input legfontosabb eszközei a C nyelv esetén a `scanf()`, az `fscanf()` és az `sscanf()`. Ezek a függvények változó paraméterszámúak. Mindegyiknek van egy ún. formátumsztring paramétere, amely meghatározza, mi a bemeneti karaktersorozattal szemben támasztott formai elvárás. A formátumsztring alapján a bemeneti adatfolyam egyes részsztringjei átalakításra kerülnek a formátumnak megfelelő adattípusra és elmentésre kerülnek a formátumsztring utáni paraméterek által meghatározott változókba. (Ezek a paraméterek tehát memóriacímek kell legyenek, vagyis pszeudo-címszerinti paraméterátadást kell használni.) Ha az előírt formátumnak nem felel meg a bemenet tartalma, akkor a beolvasás leáll és a függvény befejeződik. A függvények visszatérési értéke a sikeresen beolvasott értékek száma, tehát az az egész szám, ami megmondja, hogy a formátumparaméter utáni változók közül hányat sikerült a bemenet alapján inicializálni. Ennek segítségével dönthető el, hogy sikeres volt-e a beolvasás, és

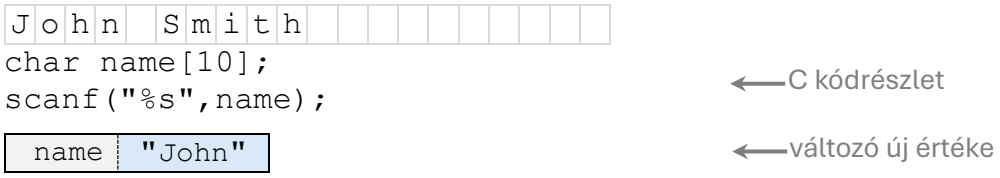
hány változó értéke származik a bemeneti adatfolyamból. A bemeneti adatfolyamot a `scanf()` esetén a billentyűzet jelenti (`stdin`), `fscanf()` esetén egy olvasásra megnyitott szöveges fájlba, `sscanf()` esetén egy sztring. A függvények definíciói (amelyek az `stdio.h` header állományban vannnek megadva) a következők:

```
int scanf (const char * formatstring, ...);
int fscanf (FILE* stream, const char * formatstring, ...);
int sscanf (char* str, const char * formatstring, ...);
```

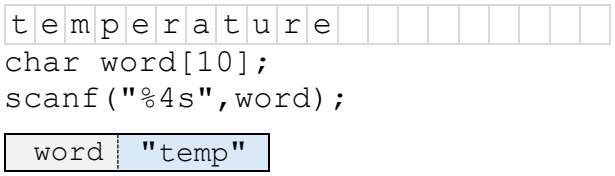
A továbbiakban egy példagyűjteményt találsz, amely összefoglalja a formátumsztringben használt speciális kifejezéseket és a beolvasás különböző lehetőségeit.

Egyszerű értékek olvasása

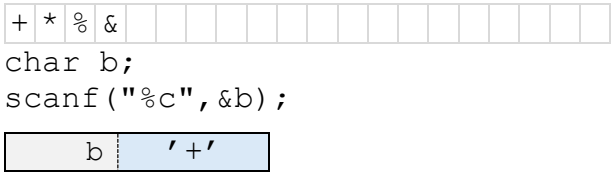
`%s`: Egy szó beolvasása egy karaktertömbbe. A beolvasás az első whitespace-ig (szóköz, tabulátor, sortörés) vagy a bemeneti adatfolyam (`stdin`) végéig tart, amelyik hamarabb fordul elő. (Ha a beolvasandó szó hosszabb, mint a tárolására megadott karaktertömb, az gondot okozhat.)



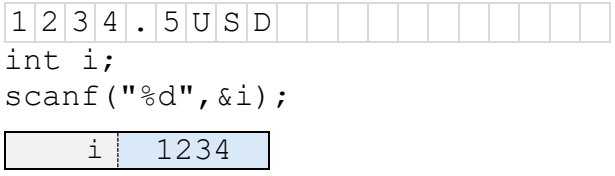
`%4s`: Egy szó, de maximum 4 karakter beolvasása egy karaktertömbbe. A hossz módosítható.



`%c`: Egy karakter (1 bájt) beolvasása.



`%d`: Egy egész számként értelmezhető karaktersorozat beolvasása, átalakítása `int` típusú értékké és eltárolása a megadott változóban.



`%2d`: Egy egész számként értelmezhető legfeljebb 2 karakter hosszúságú sztring beolvasása az input adatfolyamból, átalakítása `int` típusú értékké és eltárolása a megadott változóban. A hossz módosítható.

```
1 2 3 4 . 5 U S D
int i;
scanf("%d",&i);
```

i	12

`%f`: Egy valós (tizedestört) számként értelmezhető karaktersorozat beolvasása az input adatfolyamból, átalakítása `float` típusú értékké és eltárolása a megadott változóban. (A valós szám tudományos alakban is megadható. Dupla pontosságú lebegőpontos változó esetén a `%lf` formátumjelölés használandó.) Ha az sztring nem tartalmaz `'.'` karaktert (azaz egész számnak néz ki az érték), akkor is lebegőpontosan kerül eltárolásra.

```
1 2 3 4 . 5 U S D
float price;
scanf("%f",&price);
```

price	1234.5

`%5f`: Egy valós (tizedestört) számként értelmezhető, legfeljebb 5 karakter hosszúságú sztring beolvasása az input adatfolyamból, átalakítása `float` típusú értékké és eltárolása a megadott változóban. (A valós szám tudományos alakban is megadható.) A hossz módosítható. (A hosszúságba beleértendő az esetleges előjel és a tizedespont is.)

```
1 2 3 . 4 5 U S D
float price;
scanf("%5f",&price);
```

price	123.4

`%*c, %*d, %*f, %*s`: Egy adott típusú érték (karakter, egész szám, valós szám, sztring) értéke beolvasásra kerül, de nem lesz elmentve változóba. Gyakorlatilag egy érték kihagyása. A hossz is megadható.

```
1 s t  2 n d  3 r d
int a,b,c;
scanf("%d%*2c %d%*2c %d%*2c",&a,&b,&c);
```

a	1
b	2
c	3

Példák összetett formátum megadásra

e-mail: Egy e-mail cím beolvasása úgy, hogy külön sztringben legyen eltárolva a felhasználói név és a domain.

```
j o h n . d o e @ g m a i l . c o m
char user[20], domain[20];
scanf("%s@%s",user,domain);
```

user	"john.doe"

domain	"gmail.com"

telefonszám: Egy magyarországi vezetékes telefonszám részekre bontása. (A részek egész számként lesznek kezelve.)

```
0652512900750 16
int areacode,phonenumber,extension;
scanf("06%2d%6d%d",&areacode,&phonenumber,&extension);
```

areacode	52
phonenumber	512900
extension	75016

időjárás: Egy tabulátor szeparált hőmérséklet és páratartalom adat beolvasása külön változókba.

```
23.5°C  49%
float temperature; int humidity;
scanf("%f°C\t%d",&temperature,&humidity);
```

temperature	23.5
humidity	49

Reguláris kifejezések (regex) használata

`%[+-*/%]`: Egy kizárólag alap aritmetikai operátorokat tartalmazó sztring beolvasása. A szögletes zárójelben megadott „megengedett” karakterek kerülnek beolvasásra, ha bármilyen más karakter található a bemeneti adatfolyamban, akkor annál a beolvasás befejeződik.

```
++operator
char s[10];
scanf("%[+-*/%]",s);
```

s	"++"

`%[a-z]`: Egy kizárólag angol kisbetűket tartalmazó sztring beolvasása. A beolvasás addig tart, ameddig egy olyan karakter nem található a bemeneti adatfolyamban, amely kívül esik az `'a'` és `'z'` tartományon (ASCII 97-122). Az %[A-Z] reguláris kifejezés segítségével az angol nagy betűket (`'A'-'Z'`) tartalmazó sztring olvasható be.

```
var=2.5;
char letters[10];
scanf("%[a-z]",letters);
```

letters	"var"

`%[0-9]`: Egy kizárólag decimális számjegy karaktereket tartalmazó sztring beolvasása. (Az alábbi példában legfeljebb 16 karakter hosszú sztring beolvasása történik.)

```
1234567890123456789
char BankCardNumber[17];
scanf("%16[a-z]",BankCardNumber);
```

BankCardNumber	"1234567890123456"

`%[a-zA-z0-9_]`: Egy kizárólag alfanumerikus vagy aláhúzásjel karaktereket tartalmazó összefüggő sztring beolvasása.

```
Get_Age("John Doe");
char subroutin[20];
scanf("%[a-zA-Z0-9_]",subroutin);
```

subroutin	"Get_Age"

`%[^,]`: Egy olyan sztring beolvasása, amely a következő vessző karakterig tart. Tehát a beolvasás addig tart, ameddig a bemeneti adatfolyamban el nem ér a `^` után megadott karakterig. A beolvasás végét jelző karakter már nem kerül beolvasásra. Például a *.csv fájlok feldolgozásánál lehet hasznos. Másik hasznos példa a `%[\n]` jelölés, amely soronként történő beolvasásnál használatos.

```
John Doe,New York
char name[20];
scanf("%[^,]",name);
```

name	"John Doe"

`%*[+-]`: Előjel eldobása, azaz ha a következő karakter `'+'` vagy `'-'`, akkor az beolvasásra kerül, de nem lesz elmentve. Az alábbi példában előjeltől függetlenül egy egész abszolútérték kerül beolvasásra.

```
-123
int absval;
scanf("%*[+-]%d",&absval);
```

absval	123