Budapesti Műszaki és Gazdaságtudományi Egyetem
Közlekedés- és Járműirányítási Tanszék

Programozás C- és Matlab nyelven
C programozás kurzus
BMEKOKAM603
Console I/O, Operátorok

Dr. Bécsi Tamás

2. Előadás

Számábrázolás
Egész számok

- Számrendszerek
- Kettes számrendszer
- Számábrázolás hossza
- Negatív számok, kettes komplemens
  - $x = \begin{cases} x, ha\ x\ nem\ negatív \\ 2^n - |x|, ha\ x\ negatív \end{cases}$

Console I/O
Formátumozott adatbevitel


```
int scanf(const char *format, &var, ... );
```

- Szóközök vagy tabulátorok, amelyeket a függvény a formátum feldolgozása során figyelmen kívül hagy.
- Közönséges karakterek (nem % jel), amelyek várhatóan illeszkednek a bemeneti adatáram következő nem üreshely-karaktereihez.
- Konverziós specifikációk, amelyek a % jelből, a * opcionális hozzárendelés-elnyomó karakterből, a max. mezőszélességet meghatározó számból (opcionális), a célként megadott argumentum szélességét jelző h, l vagy L karakterből (opcionális), valamint egy konverziós karakterből tevődnek össze.

A scanf függvény konverziós karakterei

Konv kar	Arg típusa	A beolvasott adat
d	int *	decimális egész
i	int *	egész szám, ami lehet oktális (vezető nullákkal) vagy hexadecimális (vezető 0x vagy 0X karakterekkel)
o	int *	oktális egész szám (vezető nullákkal vagy azok nélkül)
u	unsigned int*	előjel nélküli decimális egész szám
x	int *	hexadecimális egész szám (a vezető 0x, ill. 0X karakterekkel vagy azok nélkül)
c	char *	karakterek. A következő bemeneti karakterek (alapfeltételezés szerint 1) elhelyezése a kijelölt mezőben. Az üres helyek átlépését (mint normális esetet) elnyomja, ha a következő nem üres karaktert akarjuk beolvastatni, akkor a %1s specifikációt kell használni
s	char *	karaktersorozat (aposztrófok nélkül). A char * mutató egy elegendően nagy karaktersorozatra mutat és a záró '\0' jelzést a beolvasás után automatikusan elhelyezi
e, f, g	float *	lebegőpontos szám, opcionális előjellel opcionális tizedesponttal és opcionális kitevővel
n	int *	az aktuális scanf hívással beolvasott karakterek száma beíródik az argumentumba. Nem történik adatbeolvasás, a konvertált tételek száma nem nő

A scanf függvény
Példák

Változók	Utasítás	Bemenet	Eredmény
`int a;`	`scanf("%d",&a);`	12	a=12
`short a;`	`scanf("%hd",&a);`	234	a=234
`short a;`	`scanf("%2hd",&a);`	234	a=23
`short a` `int n;`	`scanf("%2hd%n",&a,&n);`	234	a=23 n=2
`float f;`	`scanf("%f");`	2.3	f=2.3
`double d;`	`scanf("%lf");`	4.5	d=4.5
`char s[6];`	`scanf("%5s", s);`	hello	s= ”hello”
`char s[6];`	`scanf("%5s", s);`	hellomivan	s= ”hello”

Console Output
printf

A formátumozott adatkiviteli konverziót alapvetően a `printf` függvény különböző változatai végzik.

```
int printf(char *s, const char *format,...)
```

A függvény a `format` karaktersorozatban leírt formátum szerint átalakítja a megadott adatok értékét. A függvény a kiírt karakterek számát adja visszatérési értékként, vagy egy negatív számot, ha hiba volt.

Console Output, printf formátum karaktersorozat

A formátumot leíró karaktersorozat kétféle objektumot tartalmaz: **közönséges karaktereket**, amelyeket változtatás nélkül bemásol a kimeneti adatáramba, valamint **konverziós specifikációkat**, amelyek mindegyike az sprintf soron következő argumentumának konverzióját és kiíratását vezérli. Az egyes konverziós specifikációk a % karakterrel kezdődnek és egy konverziós karakterrel végződnek.

```
%[flags][width][.precision][length]specifier
```

Console Output, printf

%[flags][width][.precision][length]specifier

Konv kar	Arg típusa	A nyomtatás módja
d, i	int	decimális szám
o	int	előjel nélküli oktális szám vezető nullák nélkül)
x, X	int	előjel nélküli hexadecimális szám (a vezető 0x vagy 0X nélkül), a 10...15jelzése az abcdef vagy ABCDEF karakterekkel
u	int	előjel nélküli decimális szám
c	int	egyetlen karakter
s	char*	karaktersorozatból karaktereket nyomtat a '\0' végjelzésig vagy a pontossággal megadott darabszámig
f	double	[-]m.dddddd alakú decimális szám, ahol d számjegyeinek számát a pontosság adja meg (alapértelmezésben d=6)
e, E	double	[-]m.dddddde xx vagy [-]m.ddddddE xx alakú decimális szám, ahol d számjegyeinek számát a pontosság adja meg (alapértelmezésben d=6)
g, G	double	%e vagy %E alakú kiírást használ, ha a kitevő < -4 vagy >= pontosság, különben a %f alakú kiírást használja. A tizedespont és az utána következő értéktelen nullák nem íródnak ki
p	void *	mutató a géptől függő kiírási formában
n	int *	a printf függvény aktuális hívásakor kiírt karakterek száma beíródik az argumentumba. Az argumentum nem konvertálódik
%	nincs	egy % jelet ír ki

Console Output, printf
%[flags][width][.precision][length]specifier

- Jelzők (bármilyen sorrendben), amelyek módosítják a specifikációt:
- **-** mínuszjel, ami a konvertált argumentum balra igazítását írja elő a kiírási mezőben; **+** jel, ami azt írja elő, hogy a számok kiírása mindig előjellel együtt történjen; **szóközkarakter** hatására a szám elé szóköz íródik, ha az első karaktere nem előjel; **0** számkonverzió esetén azt írja elő, hogy a kiírási mezőben a szám előtti üres helyek vezető nullákkal töltődjenek fel; **#** jel a kimeneti formátum megváltoztatását írja elő. **o** esetén a kiírt első számjegy nulla lesz (oktális szám kiírása). **X** vagy **x** esetén a nem nulla szám elé 0x vagy0X íródik (hexadecimális szám kiírása), **e, E, g és G** esetén a kiírt szám mindig tartalmazza a tizedespontot és **g** vagy **G** esetén a szám végén lévő értéktelen nullák megmaradnak.

Console Output, printf
%[flags][width][.precision][length]specifier

- A kiírási mező minimális szélességét előíró szám: az átalakított argumentum legalább ilyen szélességben (vagy ha szükséges, akkor szélesebb formában) fog kiíródni. Ha az átalakított szám a megadott mezőszélességnél kevesebb karakterből áll, akkor a mező bal széle (ill. ha balra igazítás volt előírva, akkor jobb széle) helykitöltő karakterekkel fog feltöltődni. A helykitöltő karakter normális esetben szóköz, de ha 0-val való feltöltést írtuk elő, akkor nulla.

Console Output, printf
%[flags][width][.precision][length]specifier

- A pont karakter, ami elválasztja a mezőszélességet a pontosságtól.
- A pontosságot meghatározó szám, ami megadja, hogy e, E és f konverzió esetén a tizedespont után hány számjegyet kell kiírni, vagy g és G konverzió esetén minimálisan hány számjeggyel íródjon ki egy egész szám (a szükséges szélesség elérése érdekében a szám elé vezető nullák íródnak), vagy a karaktersorozatból hány karaktert kell kiírni.
- Hosszmódosító, ami h, l vagy L lehet. A h azt jelzi, hogy a megfelelő argumentum short vagy unsigned short formában nyomtatható, az l azt, hogy az argumentum long vagy unsigned long és az L pedig azt, hogy az argumentum long double.
- A mezőszélesség vagy pontosság vagy mindkettő a * jellel is megadható, és ebben az esetben a kívánt érték a következő argumentum(ok)ból, az(ok) konverziójával számítódik ki (az erre a célra használt argumentumoknak int típusúaknak kell lenni).

Console Output, printf példa

	unsinged char (245)	singed char (-11)
%d	245	-11
%u	245	4294967285
%o	365	37777777765
%x	F5	ffffffff5
%X	F5	FFFFFFF5
%5d	" 245"	
%+5d	" +245"	
%-+5d	"+245 "	
%0+5d	"+0245"	

	float 123.123456
%f	"123.123456"
%g	"123.123"
%e	"1.231235e+002"
%E	"1.231235E+002"
%+f	"+123.123456"
%014f	"0000123.123456"
%-14f	"123.123456 "
% f	" 123.123456"
%7.1f	" 123.1"
%.1f	"123.1”

Változó deklarációk

Deklarációk:

```
int i;
float f,g;
char c;
```

Inicializálás:

```
char c=a;
char szoveg[]=”szöveg”;
const int j=12;
```

Értékadás operátor

- *Értékadás:*
- `i=2;`
  Az olyan kifejezések esetén, ahol a bal oldali érték megjelenik a jobb oldalon, pld.:
- `i=i+2;`
  tömörebb formában is írhatjuk:
- `i+=2;`
  Az értékadásnak van visszatérő értéke!

2.5. Aritmetikai operátotok

A C nyelv kétoperandusú aritmetikai operátorai a +, -, * és /, valamint a % modulus operátor. Az egészek osztásakor a törtrészt a rendszer levágja, ezért van szükség a % modulus operátorra. Az x % y kifejezés az x/y egészosztás (egész) maradékát adja, és értéke nulla, ha x osztható y-nal.

A % operátor nem alkalmazható float és double típusú adatokra. Negatív operandusok esetén az egészosztás hányadosának csonkítása, valamint a modulus előjele gépfüggő, és ugyanez igaz az esetlegesen előforduló túlcsordulásra és alácsordulásra is.

Az egyoperandusú (unáris) + és - operátorok precedenciája a legmagasabb. A kétoperandusú + és - operátorok precedenciája kisebb a *, / és % precedenciájánál. Az aritmetikai operátorok mindig balról jobbra haladva hajtódnak végre (a precedencia figyelembevételével).

2.6. Relációs és logikai operátorok

A C nyelv relációs operátorai: >, >=, <, <=. Ez a sorrend egyben a precedenciájuk sorrendje is. Ezeknél eggyel alacsonyabb precedenciájúak az egyenlőség operátorok: ==, !=.
Egy relációs vagy logikai kifejezés számértéke definíció szerint 0, ha a kifejezés hamis, és 1, ha igaz.
Az && és || (ÉS illetve VAGY) operátorokkal összekapcsolt kifejezések kiértékelése balról jobbra történik, és a kiértékelés azonnal félbeszakad, ha az eredmény igaz vagy hamis volta ismertté válik.
A ! unáris (egyoperandusú) negáló operátor a nem nulla (igaz) operandust 0 értékűvé (hamissá), a 0 értékű (hamis) operandust 1 értékűvé (igazzá) alakítja.

2.8. Inkrementáló és dekrementáló operátorok

A C nyelv két szokatlan operátort használ a változók inkrementálására (eggyel való növelésére) és dekrementálására (eggyel való csökkentésére). A ++ inkrementáló operátor egyet ad az operandushoz, a -- dekrementáló operátor pedig egyet kivon belőle.

A ++ és -- szokatlan vonatkozása, hogy prefix formában (a változó előtt elhelyezve, pl. ++n) és postfix formában (a változó után elhelyezve, pl. n++) egyaránt létezik. A kétféle változat egyaránt növeli (vagy csökkenti) a változót, de a ++n a felhasználás előtt, az n++ pedig utána növeli az n értékét (a -- operátor hasonlóan működik). Ebből következően minden olyan esetben, amikor a változó értékét is felhasználjuk (nem csak a növelésre vagy csökkentésre, azaz számlálásra van szükség), a ++n és az n++ különbözik.

Ha pl. n értéke 5, akkor

```
x = n++;
```

hatására x értéke 5 lesz, amíg az

```
x = ++n;
```

hatására x értéke 6 lesz.

2.9. Bitenkénti logikai operátorok

A C nyelvben hat operátor van a bitenkénti műveletekre. Ezek az operátorok csak egész típusú adatokra, azaz char, short, int és long típusokra használhatók, akár előjeles, akár előjel nélküli változatban. Az egyes operátorok és értelmezésük a következő:

&	bitenkénti ÉS-kapcsolat
|	bitenkénti megengedő (inkluzív) VAGY-kapcsolat
^	bitenkénti kizáró (exkluzív) VAGY-kapcsolat
<<	balra léptetés
>>	jobbra léptetés
~	egyes komplemens képzés (unáris)

Operátorok összefoglalás precedencia és asszociativitás

Operátor	Asszociativitás
() [] ->	balról jobbra
! ~ ++ -- + - * & (típus) sizeof	jobbról balra
* / %	balról jobbra
+ -	balról jobbra
<< >>	balról jobbra
< <= > >=	balról jobbra
== !=	balról jobbra
&	balról jobbra
^	balról jobbra
|	balról jobbra
&&	balról jobbra
||	jobbról balra
?:	jobbról balra
= += -= *= /= %= &= ^= |= <<= >>=	balról jobbra

Köszönöm a figyelmet