



BME
Budapesti Műszaki és Gazdaságtudományi Egyetem

HAUT
Közlekedésautomatikai Tanszék

Járműfedélzeti rendszerek II.

5. előadás

Dr. Bécsi Tamás

String formázása

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

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

Az `sprintf` a kimenetet az `s` karaktersorozatba írja, majd a `'\0'` végjellel lezárja. Az `s` karaktersorozatnak elegendően hosszúnak kell lennie, hogy az eredményt tárolni tudja. A függvény visszatérésekor a kiírt karakterek számát adja (`'\0'` nélkül számolva)

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

String formázása

formátum karaktersorozat

Budapesti Műszaki és Gazdaságtudományi Egyetem Közlekedésautomatikai Tanszék

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`

String formázása

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

Budapesti Műszaki és Gazdaságtudományi Egyetem Közlekedésautomatikai Tanszék

Konvkar	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*	karaktorsorozatból 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 (alapfeltételezés szerint d=6)
e, E	double	[-]m.ddddde xx vagy [-]m.dddddE xx alakú decimális szám, ahol d számjegyeinek számát a pontosság adja meg (alapfeltételezés szerint 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

String formázása

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

Budapesti Műszaki és Gazdaságtudományi Egyetem Közlekedésautomatikai Tanszék

- 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; **szóköz** 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 vagy 0X 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.

String formázása

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

Budapesti Műszaki és Gazdaságtudományi Egyetem Közlekedésautomatikai Tanszék

- 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.

String formázása

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

Budapesti Műszaki és Gazdaságtudományi Egyetem Közlekedésautomatikai Tanszék

- 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).

String formázása

példa

Budapesti Műszaki és Gazdaságtudományi Egyetem Közlekedésautomatikai Tanszék

	unsigned char (245)	singed char (- 11)
%d	245	-11
%u	245	4294967285
%o	365	37777777765
%x	F5	ffffffff5
%X	F5	FFFFFFFF5
%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"

String formázása

példa

Budapesti Műszaki és Gazdaságtudományi Egyetem Közlekedésautomatikai Tanszék

<code>char c='a'</code> <code>char *s „szoveg”</code>	
<code>%c</code>	<code>"a"</code>
<code>%5c</code>	<code>" a"</code>
<code>%-5c</code>	<code>"a "</code>
<code>%s</code>	<code>"szoveg"</code>
<code>%-10s</code>	<code>"szoveg "</code>
<code>%10s</code>	<code>" szoveg"</code>

B3. Karakter sorozat-kezelő függvények: a `<string.h>` header

Budapesti Műszaki és Gazdaságtudományi Egyetem Közlekedésautomatikai Tanszék

- `char *strcpy(s, ct)` Az `strcpy` függvény a `ct` karakter sorozatot átmásolja az `s` karakter sorozatba, beleértve a záró `'\0'` végjelet is. A függvény visszatérési értéke `s` mutatója.
`char *strncpy(s, ct, n)` Az `strncpy` függvény a `ct`-ből `n` karaktert átmásol `s`-be és visszatér `s` mutatójával. Az `s` végét `'\0'` végjelekkel tölti fel, ha `ct` `n` karakternél rövidebb volt.
- `char *strcat(s, ct)` Az `strcat` függvény a `ct` karakter sorozatot az `s` karakter sorozat végéhez fűzi (konkatenálja) és visszatér `s` mutatójával.
- `char *strncat(s, ct, n)` Az `strncat` függvény a `ct` karakter sorozatból `n` karaktert az `s` karakter sorozat végéhez fűz, `s`-t lezárja a `'\0'` végjellel és visszatér `s` mutatójával.

Strcpy példa

```
/* strcpy example */
#include <stdio.h>
#include <string.h>
int main ()
{
    char str1[]="Sample string";
    char str2[40];
    char str3[40];
    strcpy (str2,str1);
    strcpy (str3,"copy successful");
    printf ("str1: %s\nstr2: %s\nstr3: %s\n",str1,str2,str3);
    return 0;
}
```

```
str1: Sample string
str2: Sample string
str3: copy successful
```

Strncpy példa

```
/* strncpy example */
#include <stdio.h>
#include <string.h>
int main ()
{
    char str1[] = "To be or not to be";
    char str2[40];
    char str3[40];
    /* copy to sized buffer (overflow safe): */
    strncpy ( str2, str1, sizeof(str2) );
    /* partial copy (only 5 chars): */
    strncpy ( str3, str2, 5 );
    str3[5] = '\0'; /* null character manually added */
    puts (str1);
    puts (str2);
    puts (str3);
    return 0;
}
```

```
To be or not to be
To be or not to be
To be
```

Strcat, strncat példa

```
/* strcat example */
#include <stdio.h>
#include <string.h>
int main ()
{
    char str[80];
    strcpy (str, "these ");
    strcat (str, "strings ");
    strcat (str, "are ");
    strncat (str, "concatenated. ", 5);
    puts (str);
    return 0;
}
```

these strings are concat

B3. Karaktorsorozat-kezelő

függvények: a `<string.h>` header

Budapesti Műszaki és Gazdaságtudományi Egyetem Közlekedésautomatikai Tanszék

- `int strcmp(cs, ct)` Az `strcmp` függvény összehasonlítja a `cs` karaktersorozatot a `ct` karaktersorozattal és visszatér negatív értékkel, ha `cs < ct`, nulla értékkel, ha `cs == ct` és pozitív értékkel, ha `cs > ct`.
`int strncmp(cs, ct, n)` Az `strncmp` függvény összehasonlítja a `cs` karaktersorozat legfeljebb `n` karakterét a `ct` karaktersorozattal és visszatér negatív értékkel, ha `cs < ct`, nulla értékkel, ha `cs == ct` és pozitív értékkel, ha `cs > ct`.
`char *strchr(cs, c)` Az `strchr` függvény a `c` karakter `cs`-beli első előfordulási helyének mutatójával, ill. ha `c` nem található meg `cs`-ben, akkor `NULL` értékű mutatóval tér vissza.
`char *strrchr(cs, c)` Az `strrchr` függvény a `c` karakter `cs`-beli utolsó előfordulási helyének mutatójával, ill. ha `c` nem található meg `cs`-ben, akkor `NULL` értékű mutatóval tér vissza.

Strcmp példa

```
/* strcmp example */
#include <stdio.h>
#include <string.h>
int main ()
{
    char szKey[] = "apple";
    char szInput[80];
    do {
        printf ("Guess my favourite fruit? ");
        gets (szInput);
    } while (strcmp (szKey,szInput) != 0);
    puts ("Correct answer!");
    return 0;
}
```

```
Guess my favourite fruit? orange
Guess my favourite fruit? apple
Correct answer!
```

Strncmp példa

```
/* strcmp example */
#include <stdio.h>
#include <string.h>
int main ()
{
    char str[][5] = { "R2D2" , "C3PO" , "R2A6" };
    int n;
    puts ("Looking for R2 astromech droids...");
    for (n=0 ; n<3 ; n++)
        if (strncmp (str[n], "R2xx", 2) == 0)
        {
            printf ("found %s\n", str[n]);
        }
    return 0;
}
```

```
Looking for R2 astromech droids...
found R2D2
found R2A6
```


Strchr példa

```
/* strchr example */
#include <stdio.h>
#include <string.h>
int main ()
{
    char str[] = "This is a sample string";
    char * pch;
    printf ("Looking for the 's' character in
\"%s\"...\n",str);
    pch=strchr(str,'s');
    while (pch!=NULL)
    {
        printf ("found at %d\n",pch-str+1);
        pch=strchr(pch+1,'s');
    }
    return 0;
}
```

```
Looking for the 's' character in "This is a sample string"...
found at 4
found at 7
found at 11
found at 18
```

B3. Karakter sorozat-kezelő függvények: a `<string.h>` header

Budapesti Műszaki és Gazdaságtudományi Egyetem Közlekedésautomatikai Tanszék

- `char *strstr(cs, ct)` Az `strstr` függvény visszatérési értéke a `ct` karakter sorozat `cs`-beli első előfordulásának kezdetét címző mutató, vagy `NULL`, ha a `ct` nem található meg `cs`-ben.

`size_t strlen(cs)` Az `strlen` függvény visszatérési értéke a `cs` karakter sorozat hossza.

`char *strerror(n)` Az `strerror` függvény az `n` hibaszámhoz tartozó, a gépi megvalósítástól függő hibaüzenet karakter sorozatának mutatójával tér vissza.

Strstr példa

```
/* strstr example */
#include <stdio.h>
#include <string.h>
int main ()
{
    char str[] = "This is a simple string";
    char * pch;
    pch = strstr (str, "simple");
    strncpy (pch, "sample", 6);
    puts (str);
    return 0;
}
```

This is a sample string

B3. Karaktorsorozat-kezelő függvények: a `<string.h>` header

Budapesti Műszaki és Gazdaságtudományi Egyetem Közlekedésautomatikai Tanszék

- `void *memcpy(s, ct, n)` A `memcpy` függvény a `ct`-ből `n` karaktert átmásol az `s`-be és visszatér `s` mutatójával.
- `void *memmove(s, ct, n)` A `memmove` függvény megegyezik a `memcpy` függvénnel, kivéve, hogy egymást átfedő objektumok esetén is használható.
- `int memcmp(cs, ct, n)` A `memcmp` függvény összehasonlítja a `cs` első `n` karakterét `ct`-vel. A függvény visszatérési értékei megegyeznek az `strcmp` visszatérési értékeivel. Ellentétben az `strcmp`-vel, nem áll meg `\0`-nál

Memcpy példa

```
/* memcpy example */
#include <stdio.h>
#include <string.h>
struct {
    char name[40];
    int age;
} person, person_copy;
int main ()
{
    char myname[] = "Pierre de Fermat";
    /* using memcpy to copy string: */
    memcpy ( person.name, myname, strlen(myname)+1 );
    person.age = 46;
    /* using memcpy to copy structure: */
    memcpy ( &person_copy, &person, sizeof(person) );
    printf ("person_copy: %s, %d \n", person_copy.name, person_copy.age );
    return 0;
}
```

person_copy: Pierre de Fermat, 46

Memcpy példa

```
/* memmove example */
#include <stdio.h>
#include <string.h>
int main ()
{
    char str[] = "memmove can be very useful.....";
    memmove (str+20,str+15,11);
    puts (str);
    return 0;
}
```

memmove can be very very useful.

B3. Karakter sorozat-kezelő függvények: a `<string.h>` header

Budapesti Műszaki és Gazdaságtudományi Egyetem Közlekedésautomatikai Tanszék

- `void *memchr(cs, c, n)` A `memchr` függvény visszatérési értéke a `c` karakter `cs`-beli első előfordulásának helyét címző mutató, vagy `NULL`, ha `c` nem található meg `cs` első `n` karakterében.

`void *memset(s, c, n)` A `memset` függvény elhelyezi a `c` karaktert az `s` első `n` karakterében és visszatérési értéke az `s` mutatója.

B4. Matematikai függvények: a `<math.h>` header

Budapesti Műszaki és Gazdaságtudományi Egyetem Közlekedésautomatikai Tanszék

<code>sin(x)</code>	az x argumentum szinusza;
<code>cos(x)</code>	az x argumentum koszinusza;
<code>tan (x)</code>	az x argumentum tangense;
<code>asin(x)</code>	az x argumentum árkusz szinusza, az értékkészlet a $[-\pi/2, \pi/2]$ tartomány, $x \in [-1, 1]$;
<code>acos(x)</code>	az x argumentum árkusz koszinusza, az értékkészlet a $[0, \pi]$ tartomány, $x \in [-1, 1]$;
<code>atan(x)</code>	az x argumentum árkusz tangense, az értékkészlet a $[-\pi/2, \pi/2]$ tartomány;
<code>atan2(y, x)</code>	az y/x érték árkusz tangense, az értékkészlet a $[-\pi, \pi]$ tartomány;
<code>sinh(x)</code>	az x argumentum szinusz hiperbolikusa;
<code>cosh(x)</code>	az x argumentum koszinusz hiperbolikusa;
<code>tanh(x)</code>	az x argumentum tangens hiperbolikusa;
<code>exp(x)</code>	az e^x exponenciális függvény;
<code>log(x)</code>	az x argumentum természetes alapú logaritmusa ($\ln(x)$), $x > 0$;
<code>log10(x)</code>	az x argumentum tízes alapú logaritmusa ($\lg(x)$), $x > 0$;

B4. Matematikai függvények: a `<math.h>` header

Budapesti Műszaki és Gazdaságtudományi Egyetem Közlekedésautomatikai Tanszék

<code>fabs(x)</code>	az x argumentum abszolút értéke ($ x $);
<code>ldexp(x, n)</code>	az $x \cdot 2^n$ függvény értéke;
<code>frexp(x, int *exp)</code>	a függvény az x argumentum értékét az $[1/2, 1)$ intervallumba eső normált törtrészre alakítja és ezzel az értékkel tér vissza. A 2 hatványaként értelmezett kitevő a <code>*exp</code> című változóba tárolódik. Ha $x=0$, akkor az eredmény törtrésze és kitevője egyaránt nulla lesz; result = frexp(8, &n); -> result=0.5, n=4
<code>modf(x, double *ip)</code>	az x argumentum eredeti x előjelével azonos előjelű egész- és törtrészre bontása. Az egészrész az <code>*ip</code> című változóban tárolódik, a függvény visszatérési értéke a törtrész lesz; fractpart = modf(3.14, &intpart); -> fractpart=0.14, intpart=3
<code>fmod(x, y)</code>	az x/y lebegőpontos osztás lebegőpontos maradéka, ami ugyanolyan előjelű mint x . Ha $y=0$, akkor az eredmény a gépi megvalósítástól függ. fmod(5.3, 2) -> 1.3
<code>pow(x, y)</code>	az x^y alakú hatványfüggvény, értelmezési tartomány hiba lép fel, ha $x=0$ és $y < 0$, vagy ha $x < 0$ és y értéke nem egész szám;
<code>sqrt(x)</code>	az x argumentum négyzetgyöke, $x > 0$;
<code>ceil(x)</code>	az x argumentumnál nem kisebb legkisebb egész szám, <code>double</code> típusra konvertálva;
<code>floor(x)</code>	az x argumentumnál nem nagyobb legnagyobb egész szám, <code>double</code> típusra konvertálva;

B5. Kiegészítő

rendszerfüggvények <stdlib.h>

Budapesti Műszaki és Gazdaságtudományi Egyetem Közlekedésautomatikai Tanszék

- **double atof(const char *s)** Az atof függvény az s karaktersorozat tartalmát double típusú számmá alakítja. A függvény egyenértékű az *strtod(s, (char**)NULL, 10)* függvénnel.
- **int atoi(const char *s)** Az atoi függvény az s karaktersorozat tartalmát int típusú számmá alakítja. A függvény egyenértékű az *(int)strtol(s, (char**)NULL, 10)* függvénnel.
- **long atol(const char *s)** Az atol függvény az s karaktersorozat tartalmát long típusú számmá alakítja. A függvény egyenértékű az *strtol(s, (char**)NULL, 10)* függvénnel.

B5. Kiegészítő

rendszerfüggvények <stdlib.h>

Budapesti Műszaki és Gazdaságtudományi Egyetem Közlekedésautomatikai Tanszék

- **int abs(int n)** Az abs függvény visszatérési értéke az egész típusú argumentumának abszolút értéke (egész értékként).
- long labs(long n)** A labs függvény long típusú vissztérési értéke a long típusú argumentum abszolút értéke.
- div_t div(int szaml, int nevez)** A div függvény kiszámítja az egész típusú argumentumokra felírt szaml/nevez tört hányadosát és maradékát. Az eredmény egy div_t típusú struktúra **quot** és **rem** nevű, int típusú tagjaiban tárolódik (a quot a hányadost, rem a maradékot tárolja).
- ldiv_t ldiv(long szaml, long nevez)** Az ldiv függvény kiszámítja a long típusú argumentumokra felírt szaml/nevez tört hányadosát és maradékát. Az eredmény egy ldiv_t típusú struktúra long típusú, quot és rem nevű tagjaiban tárolódik (a quot a hányadost, rem a maradékot tárolja).

Vége

Budapesti Műszaki és Gazdaságtudományi Egyetem Közlekedésautomatikai Tanszék

Köszönöm a figyelmet!