Budapesti Műszaki és Gazdaságtudományi Egyetem
Közlekedésautomatikai Tanszék

Járműfedélzeti rendszerek II.

5. előadás

Dr. Aradi Szilárd

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 a format karaktersorozatban leírt formátum szerint átalakítja a megadott adatok értékét. A függvény a kiírt karakterek számát ('\0' nélkül számolva) adja visszatérési értékként, vagy egy negatív számot, ha hiba volt.

```
char str[]="Sample string"; int x=15; float y=7.12345;
printf („szöveg: %s\negész szám: %d\nvalós: %f\n",str,x,y);
```

```
szöveg: Sample string
egész szám: 15
valós: 7.12345
```

String formázása
formátum karaktersorozat

- A formátumot leíró karaktersorozat kétféle objektumot tartalmaz
  - **közönséges karaktereket**, amelyeket változtatás nélkül bemásol a kimeneti adatáramba
  - **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

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 (alapfeltételezés szerint d=6)
e, E	double	[-]m.dddddde xx vagy [-]m.ddddddExx alakú decimális szám (normálalakban), 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
[width]

- 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]

- 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.

String formázása
[.precision][length]

- A pont karakter elválasztja a mezőszélességet a pontosságtól.
- A „**precision**” 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.
- A „**length**” a 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.
  - Az L pedig azt, hogy az argumentum long double.

String formázása példa

	unsigned char (245)	signed char (-11)
%d	245	-11
%u	245	4294967285
%o	365	37777777765
%x	F5	fffffff5
%X	F5	FFFFFFF5
%5d	" 245"	" -11"
%+5d	" +245"	" -11"
%-+5d	"+245 "	"-11 "
%0+5d	"+0245"	"-0011"

	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

char c='a' char *s "szoveg"	
%c	"a"
%5c	" a"
%-5c	"a "
%s	"szoveg"
%-10s	"szoveg "
%10s	" szoveg"

Karaktersorozat-kezelő függvények: a <string.h> header

- `char *strcpy(s, ct)` Az **strcpy** függvény a **ct** karaktersorozatot átmásolja az **s** karaktersorozatba, beleértve a **ct**-t 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** karaktersorozatot az **s** karaktersorozat végéhez fűzi (konkatenálja) és visszatér **s** mutatójával.
  `char *strncat(s, ct, n)` Az **strncat** függvény a **ct** karaktersorozatból **n** karaktert az **s** karaktersorozat végéhez fűz, **s**-t lezárja a '\0' végjellel és visszatér **s** mutatójával.

Strcpy példa


```c
/* 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


```c
/* 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. ",6);
  puts (str);
  return 0;
}
```

```
these strings are concat
```

Karaktersorozat-kezelő függvények: a <string.h> header

- `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


```c
/* 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


```
/* strncmp 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
```

Karaktersorozat-kezelő függvények: a <string.h> header

- `char *strstr(cs, ct)` Az **strstr** függvény visszatérési értéke a **ct** karaktersorozat **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** karaktersorozat hossza.

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

Strstr példa


```c
/* 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
```

Karaktersorozat-kezelő függvények: a <string.h> header

- `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** függvénnyel, kivéve, hogy egymást átfedő objektumok esetén is használható.
  `int memcmp(cs, ct, n)`A **memcmp** függvény összehasonlítja a **cs** első **n** bájtját **ct**-vel. A függvény visszatérési értékei megegyeznek az **strcmp** visszatérési értékeivel. Ellentétben az **strcmp**-vel, nem áll meg \0-nál

Memcpy példa


```c
/* 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


```c
/* 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.
```

Karaktersorozat-kezelő függvények: a <string.h> header

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

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

Matematikai függvények: a <math.h> header

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

Matematikai függvények: a <math.h> header

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

Kiegészítő rendszerfüggvények <stdlib.h>

- `double atof(const char *s)` Az **atof** függvény
  az **s** karaktersorozat tartalmát **double** típusú számmá alakítja.
- `int atoi(const char *s)` Az **atoi** függvény
  az **s** karaktersorozat tartalmát **int** típusú számmá alakítja.
- `long atol(const char *s)` Az **atol** függvény
  az **s** karaktersorozat tartalmát **long** típusú számmá alakítja.

Kiegészítő rendszerfüggvények <stdlib.h>

- `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

Köszönöm a figyelmet!