Óbudai Egyetem
Kandó Kálmán Villamosmérnöki Kar

C programozási nyelv
Magasszintű fájlkezelés II.

Dr. Schuster György

2018. január 3.

`printf` függvény

A standard kimenetre formátumozott kiíratást tesz lehetővé.

`printf` függvény

A standard kimenetre formátumozott kiíratást tesz lehetővé.Deklarációja:

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

`printf` függvény

A standard kimenetre formátumozott kiíratást tesz lehetővé.Deklarációja:

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

Paraméterei:

- **`const char *format`** az úgynevezett formátum sztring, amely formátum specifikátorokat és egyéb szöveget tartalmaz,
- `...` a többi paraméter. A C lehetővé teszi, hogy a fájl paraméter listája változhasson, ez a formátum ezt jelenti. Ide kerülnek a kiírandó adatok.

Visszatérési értéke:

- A sikeresen kiírt elemek (nem karakterek) száma.

`printf` függvény

A formátum sztring tartalmazhat formátum specifikátorokat és bármilyen más sztringként kiírandó adatot.

`printf` függvény

A formátum sztring tartalmazhat formátum specifikátorokat és bármilyen más sztringként kiírandó adatot.

A formátum specifikátor:

```
%[flags][width][.prec][length]type
```

`printf` függvény

A formátum sztring tartalmazhat formátum specifikátorokat és bármilyen más sztringként kiírandó adatot.

A formátum specifikátor:

`%[flags][width][.prec][length]type`

A formátum specifikátor mindig `%` karakterrel kezdődik.

A **`type`** az adott paraméter kiírásának értelmezését írja elő:

- **`d`**, vagy **`i int`**,
- **`u`** decimális **`unsigned`**,
- **`x`**, vagy **`X`** hexadecimális **`unsigned`**,
- **`o`** oktális **`unsigned`**,
- **`c char`**,
- **`f float`** 123.456 formában,
- **`e`**, vagy **`E float`** 1.23456e2 formában,
- **`g float`** a rövidebb formában,
- **`s`** sztring,
- **`p`** mutató.

`printf` függvény

A **`width`** paraméter a kiírás minimális szélességét adja meg. Használata nem kötelező.

Ha a kiírandó adat hosszabb, mit a **`width`** által előírt, akkor az adat nem torzul, teljesen kiíródik.

`printf` függvény

A **`width`** paraméter a kiírás minimális szélességét adja meg. Használata nem kötelező.

Ha a kiírandó adat hosszabb, mit a **`width`** által előírt, akkor az adat nem torzul, teljesen kiíródik.

A kiírt adat a rendelkezésre álló hely jobb oldalára lesz ütköztetve.

Példa:

`printf` függvény

A **`width`** paraméter a kiírás minimális szélességét adja meg. Használata nem kötelező.

Ha a kiírandó adat hosszabb, mit a **`width`** által előírt, akkor az adat nem torzul, teljesen kiíródik.

A kiírt adat a rendelkezésre álló hely jobb oldalára lesz ütköztetve.

Példa:

```
printf("\n%5d",123);
```

```
fm@dell:~$ ./width
  123
fm@dell:~$
```

`printf` függvény

A **`width`** paraméter a kiírás minimális szélességét adja meg. Használata nem kötelező.

Ha a kiírandó adat hosszabb, mit a **`width`** által előírt, akkor az adat nem torzul, teljesen kiíródik.

A kiírt adat a rendelkezésre álló hely jobb oldalára lesz ütköztetve.

Példa:

```
printf("\n%5d",123);
```

```
fm@dell:~$ ./width
  123
fm@dell:~$
```

```
printf("\n%5d",1235678);
```

```
fm@dell:~$ ./width
1235678
fm@dell:~$
```

`printf` függvény

A **`flags`** a kiírás formáját változtatja meg.
Ezek:

`printf` függvény

A **`flags`** a kiírás formáját változtatja meg.
Ezek:

- **`#`** speciális kiíratást ír elő oktális számoknál kiírja a felvezető **`0`**-t, hexadecimális számoknál kiírja a **`0x`**-et, vagy **`0X`**,
- **`0`** (nulla) a **`width`** paraméterrel együtt hatásos, ha a kiíratás mérete bagyobb, mint a kiírt érték a fent maradó területet 0-kal tölti fel. Pl.:
  `printf("\n%05d",123);`

```
fm@dell:~$ ./fnull
00123
fm@dell:~$
```

`printf` függvény

- **`-`** a **`width`** paraméterrel együtt hatásos, ha a kiíratás mérete bagyobb, mint a kiírt érték balra lesz ütköztetve. Pl.:

```
printf("\n%-5dA",123);
```

```
fm@dell:~$ ./fmin
123  A
fm@dell:~$
```

- **`szóköz`** a pozitív számok elé egy szóközt szúr be. Pl.:

```
printf("\n% 5d",123);
printf("\n% 5d",-123);
```

```
fm@dell:~$ ./fspace
123
-123
fm@dell:~$
```

- **`+`** a pozitív számok elé is kiírja az előjelet. Pl.:

```
printf("\n%+d",123);
```

```
fm@dell:~$ ./fplus
+123
fm@dell:~$
```

`printf` függvény

A `.prec` viselkedése típustól függ:

- lebegőpontos esetben a kiírandó tizedesjegyek számát határozza meg. Pl.:

```
printf("\n%.2f",123.456789);
```

```
fm@dell:~$ ./pfloat
123.46
fm@dell:~$
```

- egész jellegű változók esetén ha a `.prec` nagyobb, mint a kiírandó adat, akkor a kiírást a `.prec` mértékéig nullákkal egészíti ki. Pl.:

```
printf("\n%.5d",123);
```

```
fm@dell:~$ ./pint
00123
fm@dell:~$
```

- sztringek esetén a maximálisan kiírandó karakterek számát adja meg. Pl.: `printf("\n%.5s","abcdefg");`

```
fm@dell:~$ ./pstr
abcde
fm@dell:~$
```

`printf` függvény

A **`length`** paraméter a módosított típusok kiírását írja elő:

- **`l long`** (pl.: **`%li`**) módosított egész jellegű változók és **`double`** (pl.: **`%lf`**) kiírását végzi.
- **`ll long long`** kiíratását végzi (pl.: **`%llu`**). Nem szabványos a **`gcc`** tudja.
- **`L long double`** kiíratását végzi (**`%Lf`**).
- **`h short`** (pl.: **`%hx`**) módosított egész jellegű változók kiírását végzi.
- **`hh`** 8 bites egész jellegű változók kiírását végzi (pl.: **`%hhx`**). Nem szabványos a **`gcc`** tudja.

`printf` függvény

A `printf` működése:

```
int a=1, b=2, c=3;
    :
printf(" %d %d %d ", a , b , c );
```

A standard kimenet:

```
1 2 3
```

`printf` függvény

A `printf` működése:

```
int a=1, b=2, c=3;
    :
printf(" %d %d %d ", a , b , c );
```

A standard kimenet:

```
1 2 3
```

```
int a=1;
    :
printf("Az érték= %d ", a );
```

A standard kimenet:

```
Az érték= 1
```

`printf` függvény

A **`printf`** függvény a kiírandó változó értékeit a formátum specifikátor alapján írja ki. Ennek a következményei:

1. ha a formátum specifikátor nem felel meg a kiírandó adat típusának, akkor a kiírás hibás lesz,
2. ha kevesebb formátum specifikátor van, mint kiírandó adat, akkor csak az elöl lévő változó értékeit írja ki,
3. ha több formátum specifikátor van, mint amennyi kiírandó adat, akkor a fordítás során kapunk egy figyelmeztetést és kiír egy ”értelmetlen” adatot.

`scanf` függvény

A standard bemenetről formátumozott beolvasást tesz lehetővé.

`scanf` függvény

A standard bemenetről formátumozott beolvasást tesz lehetővé.Deklarációja:

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

`scanf` függvény

A standard bemenetről formátumozott beolvasást tesz lehetőve.Deklarációja:

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

Paraméterei:

- **`const char *format`** az úgynevezett formátum sztring, amely formátum specifikátorokat és egyéb szöveget tartalmaz,
- `...` a többi paraméter. A C lehetővé teszi, hogy a fájl paraméter listája változhasson, ez a formátum ezt jelenti. Ide kerülnek a kiírandó adatok.

Visszatérési értéke:

- A sikeresen beolvasott elemek (nem karakterek) száma.

Fontos: a beolvasandó (a paraméter listában szereplő) változóknak a címét kell megadni.

`scanf` függvény

A formátum sztring tartalmazhat formátum specifikátorokat és bármilyen más sztringet.

A formátum specifikátor:

```
%[*][width][length]type
```

`scanf` függvény

A formátum sztring tartalmazhat formátum specifikátorokat és bármilyen más sztringet.

A formátum specifikátor:

`%[*][width][length]type`

- `*` hatására a **`scanf`** függvény végigolvassa a bemenetet, de nem tölti be a kérdéses változóba,
- **`width`** a standard bemenetről maximálisan beolvasható karakterek számát adja meg,
- **`length`** teljesen megegyezik a **`printf`** függvénynél leírtakkal.

`scanf` függvény

Egy változó beolvasása egy ”white-space” karakter beütésével megtörténik, azonban a **`scanf`** függvény viszont csak egy **`ENTER`** leütésére tér vissza.

`scanf` függvény

Egy változó beolvasása egy ”white-space” karakter beütésével megtörténik, azonban a **`scanf`** függvény viszont csak egy **`ENTER`** leütésére tér vissza.

Példa:

```
#include <stdio.h>

int main(void)
   {
   int a,b,c;
   scanf("%d%d%d",&a,&b,&c);
   printf("%d%d%d\n",a,b,c);
   return 0;
   }
```

`scanf` függvény

Egy változó beolvasása egy ”white-space” karakter beütésével megtörténik, azonban a **`scanf`** függvény viszont csak egy **`ENTER`** leütésére tér vissza.

Példa:

```
#include <stdio.h>

int main(void)
    {
    int a,b,c;
    scanf("%d%d%d",&a,&b,&c);
    printf("%d%d%d\n",a,b,c);
    return 0;
    }
```

```
fm@dell:~$ ./scanfi
1 2 3
123
fm@dell:~$ ./scanfi
1 2
3
123
fm@dell:~$ ./scanfi
1
2
3
123
fm@dell:~$ ./scanfi
1
2 3
123
fm@dell:~$
```

`fprintf` függvény

Az adott megnyitott fájlba formátumozott kiíratást tesz lehetővé.

`fprintf` függvény

Az adott megnyitott fájlba formátumozott kiíratást tesz lehetővé.

Deklarációja:

```
int fprintf(FILE *fp,const char *format,...);
```

Működése teljesen megegyezik a már tárgyalt **`printf`** függvényével.

A paraméter lista első eleme az előzőleg megnyitott fájl.

`fscanf` függvény

Az adott megnyitott fájlból formátumozott beolvasást tesz lehetővé.

`fscanf` függvény

Az adott megnyitott fájlból formátumozott beolvasást tesz lehetővé.

Deklarációja:

```
int fscanf(FILE *fp,const char *format,...);
```

`fscanf` függvény

Az adott megnyitott fájlból formátumozott beolvasást tesz lehetővé.

Deklarációja:

```
int fscanf(FILE *fp,const char *format,...);
```

Működése majdnem teljesen megegyezik a már tárgyalt **`scanf`** függvényével.

A paraméter lista első eleme az előzőleg megnyitott fájl.

Különbség, hogy ”normál fájl esetén” a beolvasás végén is elegendő egy ”white-space” karakter nem kell egy ’**n**’.

`fileno` függvény

Egy magas szinten megnyitott fájl fájl leíróját adja vissza.

`fileno` függvény

Egy magas szinten megnyitott fájl fájl leíróját adja vissza.

Deklarációja:

```
int fileno(FILE *fp);
```

`fileno` függvény

Egy magas szinten megnyitott fájl fájl leíróját adja vissza.

Deklarációja:

```
int fileno(FILE *fp);
```

Paraméterei:

- **`FILE *fp`** a fájl mutató.

Visszatérési értéke:

- sikeres esetben a fájl leíró.
- hiba esetén -1, vagy másnéven **`EOF`**.

`fdopen` függvény

Egy alacsony szinten megnyitott fájl fájl mutatóját adja vissza.

`fdopen` függvény

Egy alacsony szinten megnyitott fájl fájl mutatóját adja vissza.
Deklarációja:

```
FILE *fdopen(int hnd,const char *mode);
```

`fdopen` függvény

Egy alacsony szinten megnyitott fájl fájl mutatóját adja vissza.
Deklarációja:

```
FILE *fdopen(int hnd,const char *mode);
```

Paraméterei:

- **`int hnd`** a fájl leíró,
- **`cont char *mode`** megnyitási mód, teljesen megegyezik az **`fopen`** függvénynél leírtakkal.

Visszatérési értéke:

- sikeres esetben a fájl mutató.
- hiba esetén a **`NULL`** pointer.

Példa: egy fájl dump-olása a képernyőre

```
1. #include <stdio.h>
```

Az **stdio.h** beolvasása.

Példa: egy fájl dump-olása a képernyőre

```
1. #include <stdio.h>

2. void headline(void);
3. void dump(FILE *);
```

Az **stdio.h** beolvasása.
A felhasznált függvények deklarációja.

Példa: egy fájl dump-olása a képernyőre

```
1.  #include <stdio.h>

2.  void headline(void);
3.  void dump(FILE *);

4.  int main(int argc,char *argv[])
5.    {
```

```
22.    return 0;
23.    }
```

Az **stdio.h** beolvasása.
A felhasznált függvények deklarációja.
A **main** függvény ”kerete”.

Példa: egy fájl dump-olása a képernyőre

```
1.  #include <stdio.h>

2.  void headline(void);
3.  void dump(FILE *);

4.  int main(int argc,char *argv[])
5.    {
6.    FILE  *fp;
7.    int   r;
```

```
22.   return 0;
23.   }
```

Az **stdio.h** beolvasása.
A felhasznált függvények deklarációja.
A **main** függvény ”kerete”.
Változók deklarációja.

Példa: egy fájl dump-olása a képernyőre

```
 1. #include <stdio.h>

 2. void headline(void);
 3. void dump(FILE *);

 4. int main(int argc,char *argv[])
 5.   {
 6.   FILE  *fp;
 7.   int   r;
 8.   if(argc<2)
 9.     {
10.     fprintf(stderr,"Missig parameter!\n");
11.     return -1;
12.     }
```

```
22.   return 0;
23.   }
```

Az **stdio.h** beolvasása.
A felhasznált függvények deklarációja.
A **main** függvény ”kerete”.
Változók deklarációja.
Paraméter ellenőrzés.

Példa: egy fájl dump-olása a képernyőre

```
 1. #include <stdio.h>

 2. void headline(void);
 3. void dump(FILE *);

 4. int main(int argc,char *argv[])
 5.   {
 6.   FILE  *fp;
 7.   int   r;
 8.   if(argc<2)
 9.     {
10.     fprintf(stderr,"Missig parameter!\n");
11.     return -1;
12.     }

13.   fp=fopen(argv[1],"r");









22.   return 0;
23.   }
```

Az **stdio.h** beolvasása.
A felhasznált függvények deklarációja.
A **main** függvény ”kerete”.
Változók deklarációja.
Paraméter ellenőrzés.
A kérdéses fájl megnyitása.

Példa: egy fájl dump-olása a képernyőre

```
 1. #include <stdio.h>

 2. void headline(void);
 3. void dump(FILE *);

 4. int main(int argc,char *argv[])
 5.   {
 6.   FILE  *fp;
 7.   int   r;
 8.   if(argc<2)
 9.     {
10.     fprintf(stderr,"Missig parameter!\n");
11.     return -1;
12.     }

13.   fp=fopen(argv[1],"r");

14.   if(fp==NULL)
15.     {
16.     fprintf(stderr,"File opening error!\n");
17.     return -1;
18.     }



22.   return 0;
23.   }
```

Az **stdio.h** beolvasása.
A felhasznált függvények deklarációja.
A **main** függvény ”kerete”.
Változók deklarációja.
Paraméter ellenőrzés.
A kérdéses fájl megnyitása.
A megnyitás ellenőrzése.

Példa: egy fájl dump-olása a képernyőre

```
 1. #include <stdio.h>

 2. void headline(void);
 3. void dump(FILE *);

 4. int main(int argc,char *argv[])
 5.   {
 6.   FILE  *fp;
 7.   int   r;
 8.   if(argc<2)
 9.     {
10.     fprintf(stderr,"Missig parameter!\n");
11.     return -1;
12.     }

13.   fp=fopen(argv[1],"r");

14.   if(fp==NULL)
15.     {
16.     fprintf(stderr,"File opening error!\n");
17.     return -1;
18.     }
19.   r=dump(fp);


22.   return 0;
23.   }
```

Az **stdio.h** beolvasása.
A felhasznált függvények deklarációja.
A **main** függvény ”kerete”.
Változók deklarációja.
Paraméter ellenőrzés.
A kérdéses fájl megnyitása.
A megnyitás ellenőrzése.
A **dump** függvény hívása.

Példa: egy fájl dump-olása a képernyőre

```
 1. #include <stdio.h>

 2. void headline(void);
 3. void dump(FILE *);

 4. int main(int argc,char *argv[])
 5.   {
 6.   FILE  *fp;
 7.   int   r;
 8.   if(argc<2)
 9.     {
10.     fprintf(stderr,"Missig parameter!\n");
11.     return -1;
12.     }

13.   fp=fopen(argv[1],"r");

14.   if(fp==NULL)
15.     {
16.     fprintf(stderr,"File opening error!\n");
17.     return -1;
18.     }
19.   r=dump(fp);
20.   printf("\n");

22.   return 0;
23.   }
```

Az **stdio.h** beolvasása.
A felhasznált függvények deklarációja.
A **main** függvény ”kerete”.
Változók deklarációja.
Paraméter ellenőrzés.
A kérdéses fájl megnyitása.
A megnyitás ellenőrzése.
A **dump** függvény hívása.
Csak a biztonság kedvéért.

Példa: egy fájl dump-olása a képernyőre

```
 1. #include <stdio.h>

 2. void headline(void);
 3. void dump(FILE *);

 4. int main(int argc,char *argv[])
 5.   {
 6.   FILE  *fp;
 7.   int   r;
 8.   if(argc<2)
 9.     {
10.     fprintf(stderr,"Missig parameter!\n");
11.     return -1;
12.     }

13.   fp=fopen(argv[1],"r");

14.   if(fp==NULL)
15.     {
16.     fprintf(stderr,"File opening error!\n");
17.     return -1;
18.     }
19.   r=dump(fp);
20.   printf("\n");
21.   fclose(fp);
22.   return 0;
23.   }
```

Az **stdio.h** beolvasása.
A felhasznált függvények deklarációja.
A **main** függvény ”kerete”.
Változók deklarációja.
Paraméter ellenőrzés.
A kérdéses fájl megnyitása.
A megnyitás ellenőrzése.
A **dump** függvény hívása.
Csak a biztonság kedvéért.
A fájl lezárása.

Példa: egy fájl dump-olása a képernyőre

A ”nyomtatási” fejlécet kiíró függvény.

Példa: egy fájl dump-olása a képernyőre

A ”nyomtatási” fejlécet kiíró függvény.

```
24. void headline(void)
25.   {
```

A függvény kerete.

```
35.   }
```

Példa: egy fájl dump-olása a képernyőre

A ”nyomtatási” fejlécet kiíró függvény.

```
24. void headline(void)
25.   {

26.   int   i;
```

```
35.   }
```

A függvény kerete.
`i` változó deklarációja.

Példa: egy fájl dump-olása a képernyőre

A ”nyomtatási” fejlécet kiíró függvény.

```
24. void headline(void)
25.   {

26.   int   i;

27.   printf("\n");
28.   printf("        ");
```

```
35.   }
```

A függvény kerete.
`i` változó deklarációja.
Soremelés és eltolás.

Példa: egy fájl dump-olása a képernyőre

A ”nyomtatási” fejlécet kiíró függvény.

```
24. void headline(void)
25.   {

26.   int   i;

27.   printf("\n");
28.   printf("        ");

29.   for(i=0;i<16;i++)
30.     {




33.     }



35.   }
```

A függvény kerete.
`i` változó deklarációja.
Soremelés és eltolás.
A kiíró ciklus kerete.

Példa: egy fájl dump-olása a képernyőre

A ”nyomtatási” fejlécet kiíró függvény.

```
24. void headline(void)
25.   {

26.   int   i;

27.   printf("\n");
28.   printf("        ");

29.   for(i=0;i<16;i++)
30.     {


32.     printf("%3.2x",i);

33.     }



35.   }
```

A függvény kerete.
`i` változó deklarációja.
Soremelés és eltolás.
A kiíró ciklus kerete.
A sorszámok kiírása.

Példa: egy fájl dump-olása a képernyőre

A ”nyomtatási” fejlécet kiíró függvény.

```
24. void headline(void)
25.   {

26.   int   i;

27.   printf("\n");
28.   printf("        ");

29.   for(i=0;i<16;i++)
30.     {

31.     if(i==8) printf(" ");
32.     printf("%3.2x",i);

33.     }



35.   }
```

A függvény kerete.
`i` változó deklarációja.
Soremelés és eltolás.
A kiíró ciklus kerete.
A sorszámok kiírása.
A nyolcadik sorszám után egy karakternyi eltolás.

Példa: egy fájl dump-olása a képernyőre

A ”nyomtatási” fejlécet kiíró függvény.

```
24. void headline(void)
25.   {

26.   int   i;

27.   printf("\n");
28.   printf("        ");

29.   for(i=0;i<16;i++)
30.     {

31.     if(i==8) printf(" ");
32.     printf("%3.2x",i);

33.     }

34.   printf("\n");

35.   }
```

A függvény kerete.
`i` változó deklarációja.
Soremelés és eltolás.
A kiíró ciklus kerete.
A sorszámok kiírása.
A nyolcadik sorszám után egy karakternyi eltolás.
Soremelés.

Példa: egy fájl dump-olása a képernyőre

A tulajdonképpeni dump függvény.

;

Példa: egy fájl dump-olása a képernyőre

A tulajdonképpeni dump függvény.

```
36. int dump(FILE *fp)
37.   {
```

A függvény kerete az átadott paraméterrel.

```
59.   return 0;
60.   }
```

Példa: egy fájl dump-olása a képernyőre

A tulajdonképpeni dump függvény.

```
36. int dump(FILE *fp)
37.   {
38.   char  c;
39.   int   i,j;
40.   int   r;
```

A függvény kerete az átadott paraméterrel.
A szükséges változók deklarációja.

```
59.   return 0;
60.   }
```

Példa: egy fájl dump-olása a képernyőre

A tulajdonképpeni dump függvény.

```
36. int dump(FILE *fp)
37.   {
38.   char  c;
39.   int   i,j;
40.   int   r;
41.   headline();
```

```
59.   return 0;
60.   }
```

A függvény kerete az átadott paraméterrel.
A szükséges változók deklarációja.
Az első fejléc kiírása.

Példa: egy fájl dump-olása a képernyőre

A tulajdonképpeni dump függvény.

```
36. int dump(FILE *fp)
37.   {
38.   char  c;
39.   int   i,j;
40.   int   r;
41.   headline();
42.   for(j=0;1;j++)
43.     {
```

```
58.     }
59.   return 0;
60.   }
```

A függvény kerete az átadott paraméterrel.
A szükséges változók deklarációja.
Az első fejléc kiírása.
A külső ”lap” ciklus kerete.

Példa: egy fájl dump-olása a képernyőre

A tulajdonképpeni dump függvény.

```
36. int dump(FILE *fp)
37.   {
38.   char  c;
39.   int   i,j;
40.   int   r;
41.   headline();
42.   for(j=0;1;j++)
43.     {
44.     printf(" %5.5X0  ",j);
```

```
58.     }
59.   return 0;
60.   }
```

A függvény kerete az átadott paraméterrel.
A szükséges változók deklarációja.
Az első fejléc kiírása.
A külső ”lap” ciklus kerete.
Az aktuális sor cím kiírása.

Példa: egy fájl dump-olása a képernyőre

A tulajdonképpeni dump függvény.

```
36. int dump(FILE *fp)
37.   {
38.   char  c;
39.   int   i,j;
40.   int   r;
41.   headline();
42.   for(j=0;1;j++)
43.     {
44.     printf(" %5.5X0  ",j);
45.     for(i=0;i<16;i++)
46.       {
51.       }
58.     }
59.   return 0;
60.   }
```

A függvény kerete az átadott paraméterrel.
A szükséges változók deklarációja.
Az első fejléc kiírása.
A külső ”lap” ciklus kerete.
Az aktuális sor cím kiírása.
A belső ciklus kerete.

Példa: egy fájl dump-olása a képernyőre

A tulajdonképpeni dump függvény.

```
36. int dump(FILE *fp)
37.   {
38.   char  c;
39.   int   i,j;
40.   int   r;
41.   headline();
42.   for(j=0;1;j++)
43.     {
44.     printf(" %5.5X0  ",j);
45.     for(i=0;i<16;i++)
46.       {
47.       r=fscanf(fp,"%c",&c);


51.       }


58.     }
59.   return 0;
60.   }
```

A függvény kerete az átadott paraméterrel.
A szükséges változók deklarációja.
Az első fejléc kiírása.
A külső ”lap” ciklus kerete.
Az aktuális sor cím kiírása.
A belső ciklus kerete.
Az aktuális bájt beolvasása.

Példa: egy fájl dump-olása a képernyőre

A tulajdonképpeni dump függvény.

```
36. int dump(FILE *fp)
37.   {
38.   char  c;
39.   int   i,j;
40.   int   r;
41.   headline();
42.   for(j=0;1;j++)
43.     {
44.     printf(" %5.5X0  ",j);
45.     for(i=0;i<16;i++)
46.       {
47.       r=fscanf(fp,"%c",&c);

49.       printf("%2.2hhX ",c);

51.       }









58.     }
59.   return 0;
60.   }
```

A függvény kerete az átadott paraméterrel.
A szükséges változók deklarációja.
Az első fejléc kiírása.
A külső ”lap” ciklus kerete.
Az aktuális sor cím kiírása.
A belső ciklus kerete.
Az aktuális bájt beolvasása.
Az aktuális bájt kiírása hexadecimális formában.

Példa: egy fájl dump-olása a képernyőre

A tulajdonképpeni dump függvény.

```
36. int dump(FILE *fp)
37.   {
38.   char  c;
39.   int   i,j;
40.   int   r;
41.   headline();
42.   for(j=0;1;j++)
43.     {
44.     printf(" %5.5X0  ",j);
45.     for(i=0;i<16;i++)
46.       {
47.       r=fscanf(fp,"%c",&c);

49.       printf("%2.2hhX ",c);
50.       if(i==7) printf(" ");
51.       }
```

```
58.     }
59.   return 0;
60.   }
```

A függvény kerete az átadott paraméterrel.
A szükséges változók deklarációja.
Az első fejléc kiírása.
A külső ”lap” ciklus kerete.
Az aktuális sor cím kiírása.
A belső ciklus kerete.
Az aktuális bájt beolvasása.
Az aktuális bájt kiírása hexadecimális formában.
Minden 8. kiírás után egy szóköz beszúrása.

Példa: egy fájl dump-olása a képernyőre

A tulajdonképpeni dump függvény.

```
36. int dump(FILE *fp)
37.   {
38.   char  c;
39.   int   i,j;
40.   int   r;
41.   headline();
42.   for(j=0;1;j++)
43.     {
44.     printf(" %5.5X0  ",j);
45.     for(i=0;i<16;i++)
46.       {
47.       r=fscanf(fp,"%c",&c);
48.       if(r<1) return -1;
49.       printf("%2.2hhX ",c);
50.       if(i==7) printf(" ");
51.       }
58.     }
59.   return 0;
60.   }
```

A függvény kerete az átadott paraméterrel.
A szükséges változók deklarációja.
Az első fejléc kiírása.
A külső ”lap” ciklus kerete.
Az aktuális sor cím kiírása.
A belső ciklus kerete.
Az aktuális bájt beolvasása.
Az aktuális bájt kiírása hexadecimális formában.
Minden 8. kiírás után egy szóköz beszúrása.
Ha a beolvasás véget ért, vagy hibás a függvény visszatér.

Példa: egy fájl dump-olása a képernyőre

A tulajdonképpeni dump függvény.

```
36. int dump(FILE *fp)
37.   {
38.   char  c;
39.   int   i,j;
40.   int   r;
41.   headline();
42.   for(j=0;1;j++)
43.     {
44.     printf(" %5.5X0  ",j);
45.     for(i=0;i<16;i++)
46.       {
47.       r=fscanf(fp,"%c",&c);
48.       if(r<1) return -1;
49.       printf("%2.2hhX ",c);
50.       if(i==7) printf(" ");
51.       }
52.     printf("\n");
58.     }
59.   return 0;
60.   }
```

A függvény kerete az átadott paraméterrel.
A szükséges változók deklarációja.
Az első fejléc kiírása.
A külső ”lap” ciklus kerete.
Az aktuális sor cím kiírása.
A belső ciklus kerete.
Az aktuális bájt beolvasása.
Az aktuális bájt kiírása hexadecimális formában.
Minden 8. kiírás után egy szóköz beszúrása.
Ha a beolvasás véget ért, vagy hibás a függvény visszatér.
Egy sor beszúrása.

Példa: egy fájl dump-olása a képernyőre

A tulajdonképpeni dump függvény.

```
36. int dump(FILE *fp)
37.   {
38.   char  c;
39.   int   i,j;
40.   int   r;
41.   headline();
42.   for(j=0;1;j++)
43.     {
44.     printf(" %5.5X0  ",j);
45.     for(i=0;i<16;i++)
46.       {
47.       r=fscanf(fp,"%c",&c);
48.       if(r<1) return -1;
49.       printf("%2.2hhX ",c);
50.       if(i==7) printf(" ");
51.       }
52.     printf("\n");
53.     if(j%8==0)
54.       {
55.       headline();
56.       printf("\n");
57.       }
58.     }
59.   return 0;
60.   }
```

A függvény kerete az átadott paraméterrel.
A szükséges változók deklarációja.
Az első fejléc kiírása.
A külső ”lap” ciklus kerete.
Az aktuális sor cím kiírása.
A belső ciklus kerete.
Az aktuális bájt beolvasása.
Az aktuális bájt kiírása hexadecimális formában.
Minden 8. kiírás után egy szóköz beszúrása.
Ha a beolvasás véget ért, vagy hibás a függvény visszatér.
Egy sor beszúrása.
Minden nyolcadik sor után új fejléc kiírása.

Kérdések

Mit csinál a `printf` függvény?

Kérdések

Mit csinál a **`printf`** függvény?
Formátumozott módon kiír a **`stdout`**-ra.

Kérdések

Mit csinál a **`printf`** függvény?
Formátumozott módon kiír a **`stdout`**-ra.
Mi lehet a **`printf`** függvény formátum sztringjében?

Kérdések

Mit csinál a **`printf`** függvény?
Formátumozott módon kiír a **`stdout`**-ra.
Mi lehet a **`printf`** függvény formátum sztringjében?
Formátum specifikátorok és más szöveg.

Kérdések

Mit csinál a **`printf`** függvény?
Formátumozott módon kiír a **`stdout`**-ra.
Mi lehet a **`printf`** függvény formátum sztringjében?
Formátum specifikátorok és más szöveg.
Mi jellemzi a formátum specifikátort?

Kérdések

Mit csinál a **`printf`** függvény?
Formátumozott módon kiír a **`stdout`**-ra.
Mi lehet a **`printf`** függvény formátum sztringjében?
Formátum specifikátorok és más szöveg.
Mi jellemzi a formátum specifikátort?
Minden esetben % karakterrel kezdődik és legalább egy típus azonosító van még benne.

Kérdések

Mit csinál a **`printf`** függvény?
Formátumozott módon kiír a **`stdout`**-ra.
Mi lehet a **`printf`** függvény formátum sztringjében?
Formátum specifikátorok és más szöveg.
Mi jellemzi a formátum specifikátort?
Minden esetben % karakterrel kezdődik és legalább egy típus azonosító van még benne.
Mi történik azzal az adattal a formátum sztringben, amit a **`printf`** nem tud forátum specifikátorként értelmezni?

Kérdések

Mit csinál a **`printf`** függvény?
Formátumozott módon kiír a **`stdout`**-ra.
Mi lehet a **`printf`** függvény formátum sztringjében?
Formátum specifikátorok és más szöveg.
Mi jellemzi a formátum specifikátort?
Minden esetben % karakterrel kezdődik és legalább egy típus azonosító van még benne.
Mi történik azzal az adattal a formátum sztringben, amit a **`printf`** nem tud forátum specifikátorként értelmezni?
Kiírja az **`stdout`**-ra.

Kérdések

Mit csinál a **`printf`** függvény?
Formátumozott módon kiír a **`stdout`**-ra.
Mi lehet a **`printf`** függvény formátum sztringjében?
Formátum specifikátorok és más szöveg.
Mi jellemzi a formátum specifikátort?
Minden esetben % karakterrel kezdődik és legalább egy típus azonosító van még benne.
Mi történik azzal az adattal a formátum sztringben, amit a **`printf`** nem tud forátum specifikátorként értelmezni?
Kiírja az **`stdout`**-ra.
Mi történik, ha kevesebb formátum specifikátor van, mint kiírandó adat?

Kérdések

Mit csinál a **`printf`** függvény?
Formátumozott módon kiír a **`stdout`**-ra.
Mi lehet a **`printf`** függvény formátum sztringjében?
Formátum specifikátorok és más szöveg.
Mi jellemzi a formátum specifikátort?
Minden esetben % karakterrel kezdődik és legalább egy típus azonosító van még benne.
Mi történik azzal az adattal a formátum sztringben, amit a **`printf`** nem tud forátum specifikátorként értelmezni?
Kiírja az **`stdout`**-ra.
Mi történik, ha kevesebb formátum specifikátor van, mint kiírandó adat?
A fordítás során egy figyelmeztetést küld a fordító program és csak annyi adat kerül kiírásra, ahány formátum specifikátor lett megadva.

Kérdések

Mit csinál a **`printf`** függvény?
Formátumozott módon kiír a **`stdout`**-ra.
Mi lehet a **`printf`** függvény formátum sztringjében?
Formátum specifikátorok és más szöveg.
Mi jellemzi a formátum specifikátort?
Minden esetben % karakterrel kezdődik és legalább egy típus azonosító van még benne.
Mi történik azzal az adattal a formátum sztringben, amit a **`printf`** nem tud forátum specifikátorként értelmezni?
Kiírja az **`stdout`**-ra.
Mi történik, ha kevesebb formátum specifikátor van, mint kiírandó adat?
A fordítás során egy figyelmeztetést küld a fordító program és csak annyi adat kerül kiírásra, ahány formátum specifikátor lett megadva.
Mi történik, ha több formátum specifikátor van megadva, mint kiírandó adat?

Kérdések

Mit csinál a **`printf`** függvény?
Formátumozott módon kiír a **`stdout`**-ra.
Mi lehet a **`printf`** függvény formátum sztringjében?
Formátum specifikátorok és más szöveg.
Mi jellemzi a formátum specifikátort?
Minden esetben % karakterrel kezdődik és legalább egy típus azonosító van még benne.
Mi történik azzal az adattal a formátum sztringben, amit a **`printf`** nem tud forátum specifikátorként értelmezni?
Kiírja az **`stdout`**-ra.
Mi történik, ha kevesebb formátum specifikátor van, mint kiírandó adat?
A fordítás során egy figyelmeztetést küld a fordító program és csak annyi adat kerül kiírásra, ahány formátum specifikátor lett megadva.
Mi történik, ha több formátum specifikátor van megadva, mint kiírandó adat?
A fordítás során egy figyelmeztetést küld a fordító program és a többlet helyére memóri szemet ír ki a függvény.

Kérdések

Mit csinál a **`printf`** függvény?
Formátumozott módon kiír a **`stdout`**-ra.
Mi lehet a **`printf`** függvény formátum sztringjében?
Formátum specifikátorok és más szöveg.
Mi jellemzi a formátum specifikátort?
Minden esetben % karakterrel kezdődik és legalább egy típus azonosító van még benne.
Mi történik azzal az adattal a formátum sztringben, amit a **`printf`** nem tud forátum specifikátorként értelmezni?
Kiírja az **`stdout`**-ra.
Mi történik, ha kevesebb formátum specifikátor van, mint kiírandó adat?
A fordítás során egy figyelmeztetést küld a fordító program és csak annyi adat kerül kiírásra, ahány formátum specifikátor lett megadva.
Mi történik, ha több formátum specifikátor van megadva, mint kiírandó adat?
A fordítás során egy figyelmeztetést küld a fordító program és a többlet helyére memóri szemet ír ki a függvény.
Mi történik, ha **`width`** paraméter megadott értéke kisebb, mint a kiírandó adat karakreinek száma?

Kérdések

Mit csinál a **`printf`** függvény?
Formátumozott módon kiír a **`stdout`**-ra.
Mi lehet a **`printf`** függvény formátum sztringjében?
Formátum specifikátorok és más szöveg.
Mi jellemzi a formátum specifikátort?
Minden esetben % karakterrel kezdődik és legalább egy típus azonosító van még benne.
Mi történik azzal az adattal a formátum sztringben, amit a **`printf`** nem tud forátum specifikátorként értelmezni?
Kiírja az **`stdout`**-ra.
Mi történik, ha kevesebb formátum specifikátor van, mint kiírandó adat?
A fordítás során egy figyelmeztetést küld a fordító program és csak annyi adat kerül kiírásra, ahány formátum specifikátor lett megadva.
Mi történik, ha több formátum specifikátor van megadva, mint kiírandó adat?
A fordítás során egy figyelmeztetést küld a fordító program és a többlet helyére memóri szemet ír ki a függvény.
Mi történik, ha **`width`** paraméter megadott értéke kisebb, mint a kiírandó adat karakreinek száma?
A függvény a teljes adatot kiírja.

Kérdések

Mit csinál a **`printf`** függvény?
Formátumozott módon kiír a **`stdout`**-ra.
Mi lehet a **`printf`** függvény formátum sztringjében?
Formátum specifikátorok és más szöveg.
Mi jellemzi a formátum specifikátort?
Minden esetben % karakterrel kezdődik és legalább egy típus azonosító van még benne.
Mi történik azzal az adattal a formátum sztringben, amit a **`printf`** nem tud forátum specifikátorként értelmezni?
Kiírja az **`stdout`**-ra.
Mi történik, ha kevesebb formátum specifikátor van, mint kiírandó adat?
A fordítás során egy figyelmeztetést küld a fordító program és csak annyi adat kerül kiírásra, ahány formátum specifikátor lett megadva.
Mi történik, ha több formátum specifikátor van megadva, mint kiírandó adat?
A fordítás során egy figyelmeztetést küld a fordító program és a többlet helyére memóri szemet ír ki a függvény.
Mi történik, ha **`width`** paraméter megadott értéke kisebb, mint a kiírandó adat karakreinek száma?
A függvény a teljes adatot kiírja.

Kérdések

Mit csinál a `scanf` függvény?

Kérdések

Mit csinál a `scanf` függvény?

Változókat olvas be a `stdin`-ről.

Kérdések

Mit csinál a `scanf` függvény?

Változókat olvas be a `stdin`-ről.

Hogyan adjuk meg a beolvasandó adatot a `scanf` függvénynek?

Kérdések

Mit csinál a **`scanf`** függvény?

Változókat olvas be a **`stdin`**-ről.

Hogyan adjuk meg a beolvasandó adatot a **`scanf`** függvénynek?

A kérdéses változó címét adjuk meg.

Kérdések

Mit csinál a **`scanf`** függvény?

Változókat olvas be a **`stdin`**-ről.

Hogyan adjuk meg a beolvasandó adatot a **`scanf`** függvénynek?

A kérdéses változó címét adjuk meg.

Mi történik akkor, ha a **`scanf`** függvény formátum specifikátorában vannak olyan részletek, amelyek nem formátum specifikátorok?

Kérdések

Mit csinál a **`scanf`** függvény?

Változókat olvas be a **`stdin`**-ről.

Hogyan adjuk meg a beolvasandó adatot a **`scanf`** függvénynek?

A kérdéses változó címét adjuk meg.

Mi történik akkor, ha a **`scanf`** függvény formátum specifikátorában vannak olyan részletek, amelyek nem formátum specifikátorok?

Ezeknek nincs hatása.

Kérdések

Mit csinál a **`scanf`** függvény?

Változókat olvas be a **`stdin`**-ről.

Hogyan adjuk meg a beolvasandó adatot a **`scanf`** függvénynek?

A kérdéses változó címét adjuk meg.

Mi történik akkor, ha a **`scanf`** függvény formátum specifikátorában vannak olyan részletek, amelyek nem formátum specifikátorok?

Ezeknek nincs hatása.

Mi zárja le a beolvasást **`scanf`** esetén?

Kérdések

Mit csinál a **scanf** függvény?

Változókat olvas be a **stdin**-ről.

Hogyan adjuk meg a beolvasandó adatot a **scanf** függvénynek?

A kérdéses változó címét adjuk meg.

Mi történik akkor, ha a **scanf** függvény formátum specifikátorában vannak olyan részletek, amelyek nem formátum specifikátorok?

Ezeknek nincs hatása.

Mi zárja le a beolvasást **scanf** esetén?

A \n karakter vagyis az **ENTER** billentyű.

Kérdések

Mit csinál a **`scanf`** függvény?

Változókat olvas be a **`stdin`**-ről.

Hogyan adjuk meg a beolvasandó adatot a **`scanf`** függvénynek?

A kérdéses változó címét adjuk meg.

Mi történik akkor, ha a **`scanf`** függvény formátum specifikátorában vannak olyan részletek, amelyek nem formátum specifikátorok?

Ezeknek nincs hatása.

Mi zárja le a beolvasást **`scanf`** esetén?

A **`\n`** karakter vagyis az **`ENTER`** billentyű.

Egy mező beolvasását mi zárja le **`scanf`** esetén?

Kérdések

Mit csinál a **`scanf`** függvény?

Változókat olvas be a **`stdin`**-ről.

Hogyan adjuk meg a beolvasandó adatot a **`scanf`** függvénynek?

A kérdéses változó címét adjuk meg.

Mi történik akkor, ha a **`scanf`** függvény formátum specifikátorában vannak olyan részletek, amelyek nem formátum specifikátorok?

Ezeknek nincs hatása.

Mi zárja le a beolvasást **`scanf`** esetén?

A **`\n`** karakter vagyis az **`ENTER`** billentyű.

Egy mező beolvasását mi zárja le **`scanf`** esetén?

Bármilyen ”white space” karakter.

Kérdések

Mit csinál a **`scanf`** függvény?

Változókat olvas be a **`stdin`**-ről.

Hogyan adjuk meg a beolvasandó adatot a **`scanf`** függvénynek?

A kérdéses változó címét adjuk meg.

Mi történik akkor, ha a **`scanf`** függvény formátum specifikátorában vannak olyan részletek, amelyek nem formátum specifikátorok?

Ezeknek nincs hatása.

Mi zárja le a beolvasást **`scanf`** esetén?

A \n karakter vagyis az **`ENTER`** billentyű.

Egy mező beolvasását mi zárja le **`scanf`** esetén?

Bármilyen ”white space” karakter.

Mit csinál a **`fprintf`** függvény?

Kérdések

Mit csinál a **scanf** függvény?
Változókat olvas be a **stdin**-ről.
Hogyan adjuk meg a beolvasandó adatot a **scanf** függvénynek?
A kérdéses változó címét adjuk meg.
Mi történik akkor, ha a **scanf** függvény formátum specifikátorában vannak olyan részletek, amelyek nem formátum specifikátorok?
Ezeknek nincs hatása.
Mi zárja le a beolvasást **scanf** esetén?
A \n karakter vagyis az **ENTER** billentyű.
Egy mező beolvasását mi zárja le **scanf** esetén?
Bármilyen ”white space” karakter.
Mit csinál a **fprintf** függvény?
Formátumozott (éppen úgy, ahogy a **printf**) kijratást tesz lehetővé az adott magasszinten megnyitott fájlba.

Kérdések

Mit csinál a **scanf** függvény?

Változókat olvas be a **stdin**-ről.

Hogyan adjuk meg a beolvasandó adatot a **scanf** függvénynek?

A kérdéses változó címét adjuk meg.

Mi történik akkor, ha a **scanf** függvény formátum specifikátorában vannak olyan részletek, amelyek nem formátum specifikátorok?

Ezeknek nincs hatása.

Mi zárja le a beolvasást **scanf** esetén?

A \n karakter vagyis az **ENTER** billentyű.

Egy mező beolvasását mi zárja le **scanf** esetén?

Bármilyen ”white space” karakter.

Mit csinál a **fprintf** függvény?

Formátumozott (éppen úgy, ahogy a **printf**) kijratást tesz lehetővé az adott magasszinten megnyitott fájlba.

Mit csinál az **fscanf** függvény?

Kérdések

Mit csinál a **scanf** függvény?
Változókat olvas be a **stdin**-ről.
Hogyan adjuk meg a beolvasandó adatot a **scanf** függvénynek?
A kérdéses változó címét adjuk meg.
Mi történik akkor, ha a **scanf** függvény formátum specifikátorában vannak olyan részletek, amelyek nem formátum specifikátorok?
Ezeknek nincs hatása.
Mi zárja le a beolvasást **scanf** esetén?
A \n karakter vagyis az **ENTER** billentyű.
Egy mező beolvasását mi zárja le **scanf** esetén?
Bármilyen ”white space” karakter.
Mit csinál a **fprintf** függvény?
Formátumozott (éppen úgy, ahogy a **printf**) kijratást tesz lehetővé az adott magasszinten megnyitott fájlba.
Mit csinál az **fscanf** függvény?
Formátumozott (éppen úgy, ahogy a **scanf**) beolvasást tesz lehetővé az adott magasszinten megnyitott fájlból.

Kérdések

Mit csinál a **scanf** függvény?
Változókat olvas be a **stdin**-ről.
Hogyan adjuk meg a beolvasandó adatot a **scanf** függvénynek?
A kérdéses változó címét adjuk meg.
Mi történik akkor, ha a **scanf** függvény formátum specifikátorában vannak olyan részletek, amelyek nem formátum specifikátorok?
Ezeknek nincs hatása.
Mi zárja le a beolvasást **scanf** esetén?
A \n karakter vagyis az **ENTER** billentyű.
Egy mező beolvasását mi zárja le **scanf** esetén?
Bármilyen ”white space” karakter.
Mit csinál a **fprintf** függvény?
Formátumozott (éppen úgy, ahogy a **printf**) kijratást tesz lehetővé az adott magasszinten megnyitott fájlba.
Mit csinál az **fscanf** függvény?
Formátumozott (éppen úgy, ahogy a **scanf**) beolvasást tesz lehetővé az adott magasszinten megnyitott fájlból.
Hogyan járható át a két fájlkezelés?

Kérdések

Mit csinál a **scanf** függvény?
Változókat olvas be a **stdin**-ről.
Hogyan adjuk meg a beolvasandó adatot a **scanf** függvénynek?
A kérdéses változó címét adjuk meg.
Mi történik akkor, ha a **scanf** függvény formátum specifikátorában vannak olyan részletek, amelyek nem formátum specifikátorok?
Ezeknek nincs hatása.
Mi zárja le a beolvasást **scanf** esetén?
A \n karakter vagyis az **ENTER** billentyű.
Egy mező beolvasását mi zárja le **scanf** esetén?
Bármilyen ”white space” karakter.
Mit csinál a **fprintf** függvény?
Formátumozott (éppen úgy, ahogy a **printf**) kijratást tesz lehetővé az adott magasszinten megnyitott fájlba.
Mit csinál az **fscanf** függvény?
Formátumozott (éppen úgy, ahogy a **scanf**) beolvasást tesz lehetővé az adott magasszinten megnyitott fájlból.
Hogyan járható át a két fájlkezelés?
Alacsonyról → magasra az **fdopen** függvény, magasról → alacsonyra a **fileno** függvény segítségével.

Kérdések

Mit csinál a **scanf** függvény?
Változókat olvas be a **stdin**-ről.
Hogyan adjuk meg a beolvasandó adatot a **scanf** függvénynek?
A kérdéses változó címét adjuk meg.
Mi történik akkor, ha a **scanf** függvény formátum specifikátorában vannak olyan részletek, amelyek nem formátum specifikátorok?
Ezeknek nincs hatása.
Mi zárja le a beolvasást **scanf** esetén?
A \n karakter vagyis az **ENTER** billentyű.
Egy mező beolvasását mi zárja le **scanf** esetén?
Bármilyen ”white space” karakter.
Mit csinál a **fprintf** függvény?
Formátumozott (éppen úgy, ahogy a **printf**) kijratást tesz lehetővé az adott magasszinten megnyitott fájlba.
Mit csinál az **fscanf** függvény?
Formátumozott (éppen úgy, ahogy a **scanf**) beolvasást tesz lehetővé az adott magasszinten megnyitott fájlból.
Hogyan járható át a két fájlkezelés?
Alacsonyról → magasra az **fdopen** függvény, magasról → alacsonyra a **fileno** függvény segítségével.