# Mérnöki programozás 7

Szerkesztette: dr. Vass Péter Tamás

# Függvények

**Függvény (function)**
egyedi azonosítónévvel ellátott számítási egység.
A függvények formájában kidolgozott programkódok viszonylag egyszerűbb feladatok megoldására szolgálnak.
A különböző függvények által biztosított szolgáltatások megfelelő kombinálásával az összetett feladatok megoldásai is előállíthatók.
Alkalmazásukkal sokkal áttekinthetőbb programok írhatók.
Mivel ismételten felhasználhatók egy adott programon belül, vagy különböző programokban, jelentősen megkönnyíthetik a programozási munkát.
A függvény forráskódjának részletes megadását a függvény definíciójának nevezzük.
A függvény által biztosított szolgáltatás igénybevételét a függvény hívásának nevezzük.

# Függvények

Egy függvényt mindig valamilyen másik függvényen belül hívunk meg.
Az egyetlen kivételt ez alól a main függvény jelenti, ugyanis a C nyelvű főprogram tulajdonképpen a main függvény definícióját jelenti.
A main függvény azonban számos más függvény hívását tartalmazhatja.
Azt a függvényt, ami egy másik függvény hívását tartalmazza a hívó függvénynek nevezzük.
A másik függvény pedig a hívott függvény szerepét játssza.
Egy függvény egy adott programon belül lehet egyszerre hívott függvény és hívó függvény is, mivel a különböző függvények hívásai egymásba ágyazhatók.

# Függvények

A függvények definíciójának elkészítése szempontjából két csoportba oszthatók a függvények:

- könyvtári függvények,
- felhasználó által definiált függvények.

A könyvtári függvényeket nem kell definiálni a programban, mert azok definíciója és forráskódjuk lefordítása már korábban megtörtént.

Az ilyen függvények ún. függvény könyvtárak formájában (könyvtári fájlok) vannak tárolva.

A C nyelv szabványos könyvtári függvényei minden programozó számára közös alapot jelentenek a bonyolultabb feladatok megoldásához.

Készíthetünk azonban saját magunk által fejlesztett függvényekből is függvénykönyvtárakat, ha gyakrabban van szükségünk azok hívására a programjainkban.

# Függvények

A nem könyvtári, azaz felhasználó által definiált függvényeket azonban definiálnunk kell a programjainkban.

Az egy modulból (egyetlen forrásfájlból) álló C program esetében a saját függvények definícióit a main függvény definíciója után javasolt elhelyezni.

A több modulból álló programoknál a függvények definíciói akár több forrásfájlban is elhelyezhetők. (Természetesen a fájlok tartalmát az include direktívával be kell építeni a főprogramot tartalmazó modulba.)

A függvény definíciója egyértelműen meghatározza annak működését és hívásának formáját.

Egy függvény definíciója két fő részből áll:

- a függvény fejléce,
- a függvény törzse.

# Függvények

A függvény fejléce az alábbi részekből áll balról jobbra haladva:

- a függvény visszatérési értékének (kimenő paraméter) típusa,
- a függvény neve (azonosítója),
- kerek zárójelpár.

Ha a függvény visszatérési értéke int típusú, akkor nem kötelező megadni a definícióban, de ajánlott.

A visszatérési érték nélkül definiált függvényhez ugyanis a fordítóprogram int visszatérési értéket rendel automatikusan.

Ha nincs visszatérési értéke a függvénynek, akkor void (üres) típussal kell definiálni a visszatérési értket.

A függvények névadasával kapcsolatban ugyanazok a szabályok érvényesek, mint amik a változók elnevezésére vonatkoznak.

# Függvények

A függvény nevét követő kerek zárójelpáron belül vannak felsorolva a függvény bemenőparamétereire vonatkozó információk.
Ez az ún. formális paraméterlista tartalmazza az egyes bemenő paraméterek típusait, valamint a paraméterek formális elnevezéseit.
Ha a függvénynek nincsenek bemenőparaméterei, akkor üres zárójelpárt kell alkalmazni a definíció fejlécében.

Példák függvények definíciós fejléceire:

```
double fgvnev_1(double x, double y, int n)
void fgvnev_2(float a)
int fgvnev_3()
fgvnev_4()
```

# Függvények

A függvény törzse a fejléc után következik, és kapcsos zárójelpár fogja közre. A függvény törzsén belül találhatók:

- a függvény belső változóinak definíciói (nincs mindig szükség belső változókra),
- a függvény kódját jelentő utasítások sorozata,
- a függvény kilépési pontját (ill. pontjait) meghatározó return utasítás (ill. utasítások).

Ha a függvénynek nincs visszatérési értéke (void), akkor a return utasítás után nem kell értéket vagy kifejezést megadni.

Egy függvénynek nem csak egy kilépési pont adható meg.

Példa:

```
int pozitiv(double x){
    if (x > 0)
        return 1;
    else
        return 0;
}
```

# Függvények

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

main(){
    float x;

    printf("Adjon meg egy szamot: ");
    if (scanf("%f",&x) != EOF){
        printf("A megadott szam erteke: %f",x);
        return EXIT_SUCCESS;
    }
    else{
        printf("Sikertelen adatbevitel!");
        return EXIT_FAILURE;
     }
}
```

# Függvények

A függvények sikertelen vagy sikeres végrehajtását az stdlib.h fejlécfájlban definiált szimbolikus konstansok segítségével jelezhetjük.
Az EXIT_SUCCESS értéke 0, és a sikeres végrehajtást azonosítja.
Az EXIT_FAILURE értéke 1 és a sikertelen végrehajtási ághoz rendelhető hozzá.

**A függvény deklarációja**

Ahhoz, hogy egy függvényt meghívhassunk a forráskódban, előtte deklarálnunk kell a függvényt, még a main függvény definíciója előtt.
A függvény deklarációja a függvény fejlécének egyszerűsített formájában adható meg.
A függvény deklarációs fejlécét a függvény prototípusának is nevezzük.
A deklarációs fejlécben nem kell megadni a bemeneti változók formális elnevezéseit, csak a típusait kell felsorolni.
Példa:

```
double fgvnev_1(double, double, int);
```

# Függvények

```
#include <stdio.h>
#include <stdlib.h>
int pozitiv(double);
int negativ(double);
int main(){
     double x;
     printf("Adjon meg egy szamot: ");
     if (scanf("%lf",&x) != EOF){
          if (pozitiv(x))
               printf("A szam pozitiv.");
          else if (negativ(x))
               printf("A szam negativ.");
          else
               printf("A szam nulla.");
          return EXIT_SUCCESS;
     }
     else{
          printf("Sikertelen adatbevitel!");
          return EXIT_FAILURE;
     }
}
```

# Függvények

```
int pozitiv(double a){
    if (a > 0)
        return 1;
    else
        return 0;
}

int negativ(double a){
    if (a < 0)
        return 1;
    else
        return 0;
}
```

# Függvények

Valamely függvény hívásakor, már nem a formális paraméterlistát kell alkalmazni, hanem az aktuális paramétereknek megfelelő változók azonosítóit kell a típus megadása nélkül beírni a paraméterlistába. Emiatt a függvény hívásában szereplő paraméterlistát aktuális paraméterlistának nevezzük.

A függvény hívásakor kapott bemeneti változók értékei átmásolódnak a függvény számára fenntartott memóriaterületre, és ezek az értékek felhasználására kerülnek a függvény törzsének utasításaiban.

Amikor a függvény utasításainak végrehajtása befejeződik, akkor a megfelelő return utasítás utáni kifejezés értékével tér vissza a függvény a hívást tartalmazó utasításhoz.

A hívott függvény visszatérési értékét a hívó függvényen belül fel lehet használni.

A függvény paraméterek érték szerinti átadása miatt a bemenő paraméterek értékei nem változtathatók meg közvetlenül a függvény segítségével.

# Függvények

```
#include <stdio.h>

void cserel1(int, int);

main(){
        int a=5, b=10;

        printf("a= %d\tb= %d\n",a,b);
        cserel1(a,b);
        printf("a= %d\tb= %d\n",a,b);
}

void cserel1(int x, int y){
        int s;

        s=x;
        x=y;
        y=s;
        printf("x= %d\ty= %d\n",x,y);
}
```

# Függvények

A bemenő paraméterek értékének megváltoztatása csak közvetett módon valósítható meg, amikor nem a változók értékeit veszi át a függvény, hanem a változók memóriacímeit.
Ehhez azonban mutató típusú bemenő paramétereket kell definiálni a függvény számára.

# Függvények

```c
#include <stdio.h>

void cserel2(int *, int *);

main(){
        int a=5, b=10;

        printf("a= %d\tb= %d\n",a,b);
        cserel2(&a,&b);
        printf("a= %d\tb= %d\n",a,b);
}

void cserel2(int *x, int *y){
        int s;

        s=*x;
        *x=*y;
        *y=s;
        printf("x= %d\ty= %d\n",*x,*y);
}
```

# Függvények

**Egydimenziós tömb, mint függvény bemenő paraméter**

Tömböt, ill. annak elemeit nem lehet érték szerint átadni egy függvénynek bemenő paraméter formájában.
A tömb kezdőcímét azonban mutató típusú bemenő paraméternek át lehet adni, és így a tömb elemein műveleteket lehet végrehajtani a függvény törzsében.
A tömb memóriaterületének kezdőcímén kívül azonban meg kell adni a tömb méretét is, mint bemenő paraméter, mivel a kezdőcím semmiféle információt nem tartalmaz a tömb méretére vonatkozóan.

# 1. megoldás

```
#include <stdio.h>

void vektorbeolvas(float *, int);
float skalarszorzat(float *, float *, int);

main(){
   const int n=5;
   float a[n], b[n];

   printf("Adja meg az elso vektor koordinatait!\n");
   vektorbeolvas(a,n);
   printf("Adja meg a masodik vektor koordinatait!\n");
   vektorbeolvas(b,n);
   printf("A ket vektor skalarszorzata: %f",
skalarszorzat(a,b,n));
}
```

# 1. megoldás

```
void vektorbeolvas(float *v, int meret){
    int i;
    for (i=0; i<meret; i++){
        printf("%d. koordinata erteke: ", i+1);
        scanf("%f", v+i);
    }
}

float skalarszorzat(float *x, float *y, int meret){
    int i;
    float s=0;

    for (i=0; i<meret; i++)
        s=s+*(x+i)**(y+i);
    return s;
}
```

# 2. megoldás

```
#include <stdio.h>

void vektorbeolvas(float [], int);
float skalarszorzat(float [], float [], int);

main(){
    const int n=5;
    float a[n], b[n];

    printf("Adja meg az elso vektor koordinatait!\n");
    vektorbeolvas(a,n);
    printf("Adja meg a masodik vektor koordinatait!\n");
    vektorbeolvas(b,n);
    printf("A ket vektor skalarszorzata: %f",
skalarszorzat(a,b,n));
}
```

# 2. megoldás

```
void vektorbeolvas(float v[], int meret){
    int i;

    for (i=0; i<meret; i++){
        printf("%d. koordinata erteke: ", i+1);
        scanf("%f", &v[i]);
    }
}

float skalarszorzat(float x[], float y[], int meret){
    int i;
    float s=0;

    for (i=0; i<meret; i++)
       s=s+x[i]*y[i];
    return s;
}
```

# Szöveges állományok kezelése

Az adatállományokat tartalmuk szerint két nagy csoportba oszthatjuk:

- szöveges állományok,
- bináris kódolású állományok.

A szöveges állományok valamilyen karakterkód tábla szerint kódolt adatokat tartalmaznak.

A szöveges állományok tartalma általában közvetlenül megjeleníthető és olvasható egyszerű szövegszerkesztő program segítségével.

A szöveges fájlok sorokból épülnek fel, a sorok végét speciális vezérlőkarakterek jelzik ('\n').

A fájl legutolsó sorát szintén speciális vezérlő karakter jelzi (EOF = end of file).

A szöveges fájlok kezelésének legfontosabb műveletei

- az adatok beolvasása fájlból,
- és az adatok kiírása fájlba.

# Szöveges állományok kezelése

A szöveges fájlok kezelésének lépései a következők:

- a fájlt azonosító mutató definiálása,
- a fájl megnyitása,
- olvasás a fájlból és/vagy írás a fájlba,
- a fájl lezárása.

**A fájl mutató definiálása**

A fájlokat adatfolyamként kezeljük a C nyelvű programjainkban. Minden egyes fájlt, mint adatfolyamot, egy file mutató alkalmazásának segítségével érhetünk el.

A fájl mutató szerkezetét az stdio.h fejlécfájl deklarálja, ezért a fájlkezeléshez ezt az állományt mindig be kell építeni a C forrásprogramunkba.

A fájl mutató definíciója:

```
FILE *fp;
```

# Szöveges állományok kezelése

A definícióban a `FILE *` a fájlmutató típusát jelzi, az `fp` pedig egy azonosítónevet határoz meg az adott fájlmutató számára.
Ha több szöveges adatfájllal végzünk műveleteket a programunkban, akkor minden egyes fájlhoz definiálnunk kell egy egyedi azonosítójú fájlmutatót.

```
FILE *fp1, *fp2;
```

A file megnyitása
A háttértáron lévő fájlhoz csak akkor tudunk hozzáférni, ha azt megnyitjuk.
A file megnyitásakor:

- kapcsolatot teremtünk a fájlmutató és a fájl között,
- valamint meghatározzuk a hozzáférés módját.

A fájl megnyitására szolgáló szabványos könyvtári függvény az fopen.

# Szöveges állományok kezelése

Az fopen függvény fejléce:
FILE * fopen(const char *fajlnev, const char *mod);

A *fajlnev* bemenőparaméterben adjuk át a függvénynek a megnyitni kívánt függvény nevét egy sztring (karakterlánc formájában).
Ha a fájl nem ugyanabban a mappában helyezkedik el, mint a C programunk, akkor a fájl neve előtt a teljes elérési útvonalat is meg kell adni a meghajtó azonosítójával együtt.
Példa:
"C:\\Users\\Public\\Documents\\adatok.txt"
A *mod* bemenőparaméterben adjuk meg az elérés módját egy sztring formájában.
A *mod* paraméter lehetséges értékei:

| | |
|---|---|
| "r" | létező fájl megnyitása adatok beolvasásra, |
| "w" | új fájl megnyitása adatok kiírására, ha a fájl már létezik, akkor a tartalma elvész, |

# Szöveges állományok kezelése

A *mod* paraméter lehetséges értékei:

| | |
|---|---|
| "a" | fájl megnyitása adatok hozzáfűzésére, ha a fájl nem létezik még, akkor létrehozza a program, |
| "r+" | létező fájl megnyitása írásra és olvasásra, |
| "w+" | új fájl megnyitása írásra és olvasásra, ha a fájl már létezik, akkor a tartalma elvész, |
| "a+" | létező fájl megnyitása a fájl végén végzett olvasásra és írásra, ha a fájl még nem létezik, akkor a program létrehozza a fájlt. |

A megnyítási módok után fűzött *t* betű a text fájltípust jelzi, míg a *b* betű a bináris fájltípust adja meg.

Például:

"r+t"

Ha nem adjuk, meg a fájltípus betűjelét, akkor a fordító text típusú fájlra vonatkoztatja a fájl megnyitási műveletet.

# Szöveges állományok kezelése

Az fopen függvény visszatérési értéke egy FILE * típusú mutató, ami sikeres fájl megnyitás esetén a fájlhoz rendelt struktúra memóriacímét tartalmazza.
Ha nem sikerült a fájl megnyitása, akkor a visszatérési érték NULL lesz, azaz a fájlmutató nem tartalmaz érvényes memóriacímet.

**Fájlkezelési műveletek:**
A fájl sikeres megnyitása után már írhatjuk olvashatjuk annak tartalmát.
A szöveges fájlok kezelésénél alkalmazható szabványos könyvtári függvények fejlécei az alábbiak.
Egyetlen karakter beolvasására szolgáló függvény:

```
int fgetc(FILE *fajlmutato);
```

A függvény visszatérési értéke a beolvasott karakter kódja. Ha a beolvasás a fájl végéhez érkezik, akkor a függvény EOF értékkel tér vissza a hívó függvényhez.

# Szöveges állományok kezelése

Egyetlen karakter kiírására szolgáló függvény:

```
int fputc(int c, FILE *fajlmutato);
```

A c karakterkódú karaktert kiírja a fájlmutató által azonosított fájlba. Sikeres művelet esetén a visszatérési érték a karakter kódja, egyébként pedig EOF.

Sztring olvasása fájlból:

```
char *fgets(char *s, int n, FILE *fajlmutato);
```

a fájlmutatóval azonosított fájlból a soron következő legfeljebb n-1 hosszúságú sztringet beolvassa az s karaktertömbbe. A tömb egy '\0' karakterrel egészül ki. A visszatérési értéke sikeres végrehajtás esetén az s karaktertömb címe, hiba és fájlvége esetén NULL.

# Szöveges állományok kezelése

Sztring kiírása fájlba:

```
int fputs(const char *s, FILE *fajlmutato);
```

Kiírja az s karaktersorozatot a fájlmutatóval azonosított fájlba. Sikeres végrehajtás esetén nem negatív értékkel, hiba esetén EOF-el tér vissza a hívó függvényhez.

Formázott adatok beolvasása fájlból:

```
int fscanf(FILE *fajlmutato, const char *format
…);
```

A format sztringben megadott formátumú adatokat beolvassa a fájlmutatóval azonosított fájlból a paraméterlistában megadott memória címekre. Sikeres végrehajtás esetén a beolvasott adatok számával tér vissza, fájlvége és egyéb hiba esetén EOF a visszatérési értéke.

# Szöveges állományok kezelése

Formázott adatok kiírása a fájlba:

```
int fprintf(FILE *fajlmutato, const char *format
…);
```

A format sztringben megadott formátumú adatokat kiírja a fájlmutatóval azonosított fájlba a paraméterlistában megadott változókból. Sikeres végrehajtás esetén a kiírt karakterek számával tér vissza, hiba esetén a visszatérési értéke egy negatív szám.

**Fájl lezárása**

Ha elvégeztük a szükséges adatbeviteli és kiviteli műveleteket, akkor még a program befejezése előtt le kell zárni a megnyitott állományt, ill. állományokat.

Az fp fájlmutatóval azonosított állomány lezárása az alábbi szabványos könyvtári függvénnyel történik:

```
fclose(fp);
```

# Szöveges állományok kezelése

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

main(){
        FILE *fp;
        int i;
        float t;

        fp = fopen("adatok.txt", "wt");
        if (fp==NULL){
                printf("Sikertelen fajlmegnyitas!");
                exit(-1);
        }
        for (i=0; i<10; i++){
                t=0.2*i;
                fprintf(fp,"%f\t%f\n", t, sin(t));
        }
        printf("A fajlmuvelet sikeres volt.");
        fclose(fp);
}
```

# Szöveges állományok kezelése

```c
#include <stdio.h>
#include <stdlib.h>

main(){
        FILE *fp;
        float t, sint;

        fp = fopen("adatok.txt", "rt");
        if (fp==NULL){
                printf("Sikertelen fajlmegnyitas!");
                exit(-1);
        }
        while(!feof(fp)){
                fscanf(fp,"%f\t%f\n",&t,&sint);
                printf("%f\t%f\n",t,sint);
        }
        printf("A fajlmuvelet sikeres volt.");
        fclose(fp);
}
```