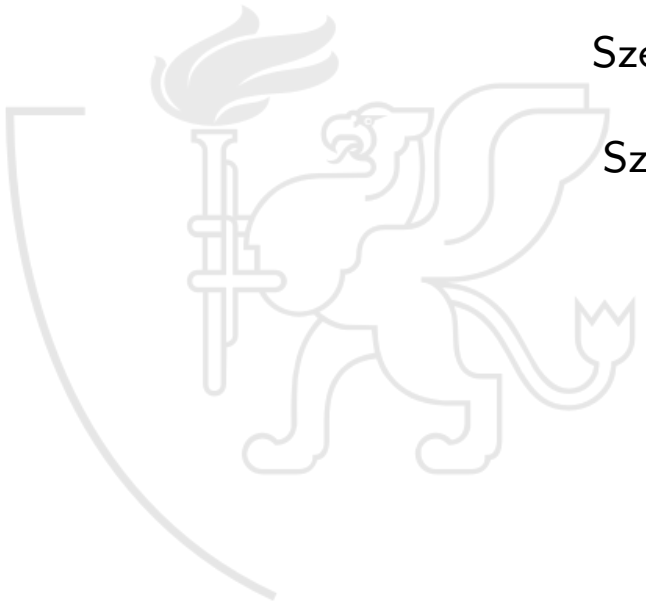
Programozás Alapjai

Dr. Gergely Tamás
Dr. Jász Judit

Szegedi Tudományegyetem
Informatikai Intézet
Szoftverfejlesztés Tanszék



2023

(v0914)

Tartalom

Tartalom

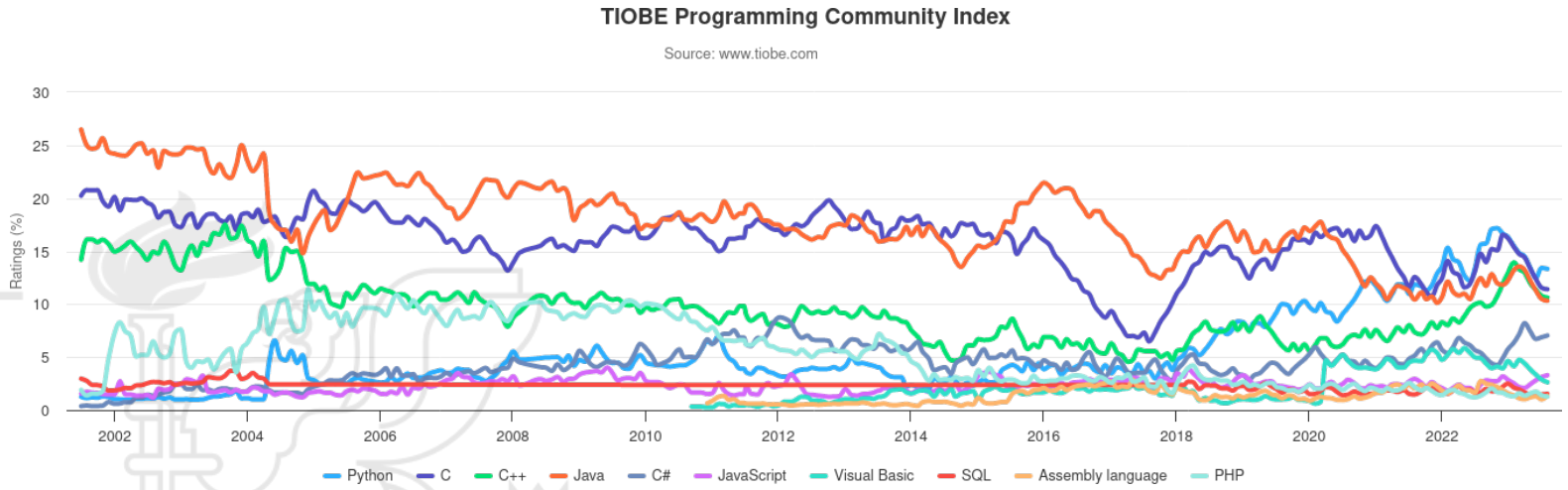
A C nyelv

Bevezetés

- Először gyakorlatias példákon keresztül áttekintjük a C nyelvet, kifejezetten a nyelvi megvalósításra koncentrálva.
  - A cél az, hogy a gyakorlatokon minél előbb el lehessen kezdeni a C nyelv gyakorlását.
- Azután megnézzük, hogyan kell programozni, áttekintjük az elméletet és leásunk a C programozási nyelv mélyére.

Miért a C programozási nyelv?

- A 2017-es év programozási nyelve*
- 2023 augusztusában 11.4%-kal a C volt a második legnépszerűbb programozási nyelv. (A 13.3%-os Python mögött, a harmadik a 10.6%-os C++, a negyedik a Java 10.3%-kal.)



(2023 augusztus) https://www.tiobe.com/tiobe-index/

*https://prog.hu/hirek/4898/nem-talalod-ki-melyik-lett-a-2017-es-ev-programozasi-nyelve

Miért népszerű?

- Magas szintű konstrukciói vannak.
- Alacsony szintű tevékenységeket is kezelni tud.
- Hatékony programot készítenek a fordítók.
- A számítógépek sok változatán találunk C fordítót.

A C nyelv

Történelem [1/2]

- UNIX fejlesztése 1969 körül az AT&T Bell Laboratóriumában.
  - Az első változat Assembly nyelven DEC PDP-7 gépen készült.
  - Assembly nyelven nehézkes fejlesztés, sok a hiba. Igény volt egy új nyelvre, ami egyszerre hardverközeli és magasszintű.
- A „B” nyelvet 1970 körül fejlesztette ki Ken Thomson.
  - Közvetlenül a BCPL nyelvből származik.
  - Legfőbb jellemzője, hogy típustalan.
  - A UNIX fejlesztése ezen a nyelven folytatódik.
- A „C” nyelvet 1973 körül fejleszti ki Ken Thompson és Dennis Ritchie.
  - A UNIX rendszert portolják PDP-11-re, de a „B” nyelv erre kevésbé alkalmas.
  - Ezért a „B” nyelv alapjain elkészítik a „C” nyelvet.
  - A UNIX operációs rendszert átírták „C”-re.

A C nyelv

Történelem [2/2]

- Szabványosítás
  - 1978: Brian Kernighan és Dennis Ritchie – „K&R C” kvázi standard
  - 1989-90: ANSI C, C89, ISO/IEC 9899:1990, C90
    - **A**merican **N**ational **S**tandards **I**nstitute
    - **I**nternational **O**rganization for **S**tandardization
    - **I**nternational **E**lectrotechnical **C**ommission
  - 1999: ISO/IEC 9899:1999, C99 (C99)
  - 2011: ISO/IEC 9899:2011, C11 (C11)
  - 2018: ISO/IEC 9899:2018, C18 (C18)
- A C nyelv és a UNIX összefonódása
  - C shell (csh): parancsértelmező C-szerű szintaxissal.
  - manual oldalak programokhoz: cc, gcc, ld, ...
  - manual oldalak C nyelvhez (3-as szekció, `man 3 ...`): printf, scanf, ...
- 2017-ben a C lett az év programozási nyelve!
  - `https://prog.hu/hirek/4898/nem-talalod-ki-melyik-lett-a-2017-es-ev-programozasi-nyelve`

A C nyelv

Tulajdonságok

- A C egy nagyon kicsi nyelv. Szintaxisa a K&R-ben csupán néhány oldal.
- A nyelv érzékeny a kis- és nagybetűkre!
- Nincs beépítve
  - I/O kezelés
  - Sztring kezelés
  - Matematikai függvények
- Gazdagon kínál standard függvénykönyvtárakat
- Függvényhívások széleskörű használata
- A típus hanyagolása
- Strukturált nyelv
- Alacsony szintű programozás olvasható elérése
- Pointer széleskörű használata
  - memória, tömb, struktúra, függvény

Segédanyagok C nyelvhez

- C programozói referenciák
  - Első Google találatok:
    - `http://en.cppreference.com/w/`
    - `http://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html`
    - `http://www.acm.uiuc.edu/webmonkeys/book/c_guide/`
    - és még sokan mások
  - GCC dokumentáció
    - `http://gcc.gnu.org`

C fejlesztői környezetek

- Microsoft Visual Studio
  - Integrált programfejlesztői rendszer Windows alá
- GNU Compiler Collection
  - `gcc` fordító program (GNU C Compiler)
  - Linux és Windows alatt is elérhető
- LLVM Compiler Infrastructure
  - `clang` fordító program (LLVM C frontend)
  - Használatban nagyrészt `gcc` kompatibilis
- „Fapados” környezetek
  - Text alapú szövegszerkesztő (vi, mcedit, gedit, kate, geany, subl) + parancssori fordító
- Integrált fejlesztői környezetek (IDE)
  - anjuta, eclipse, dev c++, codeblocks, codelite, clion

Minimális C program [1/1]

minimalis.c [1–6]

```c
/* Minimális C program, ami nem csinál semmit
 * 1998. Február 26.  Dévényi Károly, devenyi@inf.u-szeged.hu
 */

main() {
}
```

Hello Világ! [1/1]

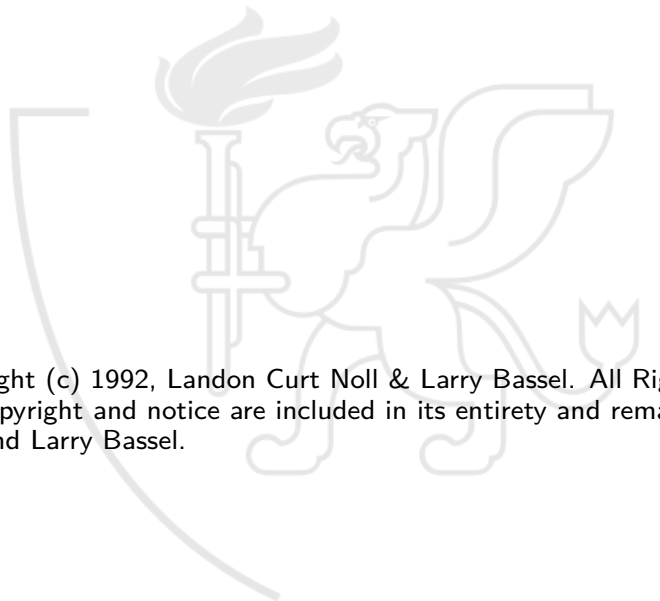
hello.c [1–10]

```c
/* Helló Világ!
 * 2004. November 3.  Gergely Tamás, gertom@inf.u-szeged.hu
 */

#include <stdio.h>

int main() {
    printf("Helló␣Világ!\n");
    return 0;
}
```

Hol vagyunk? [1/1]

where.c [1–12]

```
            main(l
       ,a,n,d)char**a;{
  for(d=atoi(a[1])/10*80-
 atoi(a[2])/5-596;n="@NKA\
CLCCGZAAQBEAADAFaISADJABBA^\
SNLGAQABDAXIMBAACTBATAHDBAN\
ZcEMMCCCCAAhEIJFAEAAABAfHJE\
TBdFLDAANEfDNBPHdBcBBBEA_AL\
␣H␣E␣L␣L␣O,␣␣␣␣W␣O␣R␣L␣D!␣"
    [l++-3];)for(;n-->64;)
       putchar(!d+++33^
            l&1);}
```



. . . és a futásának eredménye

```
$ ./where 46 20
                !!!!!!!!!!! !!!             !!!    !!!!!!!
!  !!!!!!!!!!!!!!!!!   !!!!! !     !       !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
  !!!!!!!!!!!!!!!!!! !!!!           !   !! !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
           !!!!!!!!!!!!!!            !!!!"!! !!!!!!!!!!!!!!!!!! !!     !
              !!!!!!!!!             !! !    !!!!!!!!!!!!!!!!!!!!   !
     !          !!!!   !             !!!!!!!!!!!!!!!!!!!!!!!!!!!!
                  !!!!!             !!!!!!!!!!!!!!    !!!    !!! !
                       !!!!!         !!!!!!!!!!!       !      ! !  !
                       !!!!!!!!          !!!!!                    !!
                        !!!!!!             !!!! !               !!!!!
                          !!!!               !!                !!!!!!!!
                          !!                                    !!   !!      !
                          !
$
```

Forgó fánk [1/1]

donut.c [1–21]

```
1               k;double sin()
2           ,cos();main(){float A=
3         0,B=0,i,j,z[1760];char b[
4       1760];printf("\x1b[2J");for(;;
5    ){memset(b,32,1760);memset(z,0,7040)
6    ;for(j=0;6.28>j;j+=0.07)for(i=0;6.28
7   >i;i+=0.02){float c=sin(i),d=cos(j),e=
8   sin(A),f=sin(j),g=cos(A),h=d+2,D=1/(c*
9   h*e+f*g+5),l=cos      (i),m=cos(B),n=s\
10  in(B),t=c*h*g-f*        e;int x=40+30*D*
11  (l*h*m-t*n),y=            12+15*D*(l*h*n
12  +t*m),o=x+80*y,          N=8*((f*e-c*d*g
13   )*m-c*d*e-f*g-l        *d*n);if(22>y&&
14   y>0&&x>0&&80>x&&D>z[o]){z[o]=D;;;b[o]=
15   ".,-~:;=!*#$@"[N>0?N:0];}}/*#****!!-*/
16    printf("\x1b[H");for(k=0;1761>k;k++)
17     putchar(k%80?b[k]:10);A+=0.04;B+=
18       0.02;}}/*****##########!!=;:~
19         ~::==!!!**********!!!==::-
20           .,~~;;;========;;;:~-.
21               ..,--------,*/
```

Fordítás Linux rendszerben

- C program fordítása gcc fordítóval, és futtatás.

```
$ gcc program.c
$ ./a.out
```

- Jobb megoldás, ha a programnak nevet adunk.

```
$ gcc -o program program.c
$ ./program
```

- Ha matematikai függvényeket használunk, szükségünk lesz egy plusz kapcsolóra.

```
$ gcc -o program program.c -lm
$ ./program
```

- Tanuláshoz, gyakorláshoz nem árt, ha a fordító minden problémásabb pontról tájékoztat.

```
$ gcc -Wall -o program program.c
$ ./program
```

Tartalom

Pénzváltás

penzvalto.c (v1.0) [1–16]

```c
/* Pénzváltás rögzített árfolyamon.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2022. Augusztus 30. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

int main() {
    double forint, euro;
    printf("Hány forintot váltsunk? ");
    scanf("%lf", &forint);
    euro = forint / 384.347;                                    /* 2023-09-14 */
    printf("%lf HUF = %lf EUR\n", forint, euro);
    return 0;
}
```

Pénzváltás

A probléma

- Van valamennyi pénzünk forintban.
- Mennyi Eurót tudunk vásárolni érte?

Pénzváltás

penzvalto.c (v0.1) [1–9]

```c
/* Pénzváltás rögzített árfolyamon.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Augusztus 31. Gergely Tamás, gertom.inf.u-szeged.hu
 */

int main() {
    return 0;
}
```

Egyszerű ki- és bevitel

`printf()`, `scanf()`

- Magának a C nyelvnek nem része a ki- és bevitel, de a fordítókörnyezetekben egységes módon meg van valósítva.
- A ki- és bevitel használatához szükségünk van az

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

  sorra a program elején.
- Ezek után használhatjuk az alábbi két függvényt:
  - `scanf(...)`: a bemenetről tudunk olvasni
  - `printf(...)`: a kimenetre tudunk írni
- Egyelőre anélkül, hogy a két függvényt részletesen elmagyaráznánk megmutatjuk, hogyan lehet `int` illetve `float` vagy `double` típusú értékek beolvasására, valamint ugyanilyen típusú értékek és tetszőleges szöveg kiíratására használni őket.

Egyszerű ki- és bevitel

`int`, `float`, `double`

- Mindkét függvény első paramétere egy úgynevezett *formátumsztring*. Ez tartalmazza a kiírandó / beolvasandó szöveget, és tartalmazhat úgynevezett *konverziós előírásokat*.

`%d`	`int` típusú egész érték (decimális alakban)
`%f`	`float` típusú valós érték
`%lf`	`double` típusú valós érték

- A konverziós előírás határozza meg, hogy hogyan kell a további paramétereket kiírni / beolvasni.

```
printf("Hello␣world\n");
printf("Pi␣értéke␣kb.␣%lf\n", 3.14159265358979323846);
printf("%d␣+␣%d␣=␣%d\n", 2, 3, 2+3);
scanf("%d", &egesz);
scanf("%f␣%f", &valos1, &valos2);
```

Egyszerű ki- és bevitel

Típusegyeztetés

- Nagyon fontos, hogy a beolvasandó értékek illetve a kiírandó kifejezések számát és típusát sorban és pontosan adjuk meg.
- Egy `double` típusú kifejezés vagy változó esetén tehát akkor sem a `%d` kombinációt használjuk, ha tudjuk, hogy maga az érték egész, és `int` típusú kifejezés illetve változó esetén sem használhatjuk a `%lf` konverziós specifikációt.
- Az alábbi példák tehát hibásak:

```
printf("%d", 10.0);
printf("%lf", 10);
```

Sztring literálok

- A sztringek (szövegek) C-ben tulajdonképpen karaktersorozatok, egymás után írt `char` típusú értékek egyetlen egységként kezelve.
- Sztring literálokat idézőjelek között lehet megadni.
- Egymástól csak whitespace karakterekkel elválasztott sztring literálokat a fordító összefűzi, és egyetlen értékként kezeli.
- A sztringről mint típusról később lesz szó.

```
"Helló␣Világ!\n"
"Hány␣forintot␣váltsunk?"
"%lf"
"%lf␣HUF␣=␣%lf␣EUR\n"
"Egybe" "írva␣és␣külön␣is," "␣ez␣egyetlen␣sztring␣lesz."
```

Pénzváltás

penzvalto.c (v0.2) [1–12]

```c
/* Pénzváltás rögzített árfolyamon.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Augusztus 31. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

int main() {
    printf("Hány forintot váltsunk? ");
    return 0;
}
```

Egy C program felépítése

Egyszerűbb program

- Egy egyszerű C program így néz ki:

```c
/* A program adatai */

#include <stdio.h>

int main() {

    Változódeklarációk

    Utasítások

    return 0;
}
```

Deklaráció

Deklaráció *Egy programkomponens deklarációja egy azonosító (név) hozzárendelése az adott komponenshez.*

- Ezzel az azonosítóval hivatkozhatunk a program további részében az adott komponensre (adatra, műveletre, értékre, adattípusra).
- A program egy adott pontján csak azok a komponensek használhatók (hivatkozhatók), amelyeket e pontot megelőzően már deklaráltunk, ellenkező esetben fordítási hiba lép fel. (Egyes fordítók ennél megengedőbbek lehetnek.)

Adattípus

Adattípus *Az adattípus a programnak egy olyan komponense, amely két összetevője, az értékhalmaz és az értékhalmaz elemein végezhető műveletek által meghatározott.*

- Minden adattípus vagy elemi, vagy más adattípusokból képzett összetett adattípus.

A C nyelv elemi adattípusai

Karakter `char`

- Egy karakter (betű, szám, írásjel, ...) ábrázolására való.

Egész `short int`, `int`, `long int`

- Egész számok ábrázolására való típusok, értékkészletükben különböznek.

Valós `float`, `double`

- Valós számok ábrázolására való típusok, értékkészletükben, pontosságukban különböznek.

Logikai `_Bool`

- A C nyelvnek C99 előtt nem volt része a logikai `_Bool` adattípus, de logikai értéket adó műveletek már akkor is voltak. A C99 szabvány óta létező `stdbool.h` header fájl pedig tartalmazza a `true` és `false` értékekkel rendelkező `bool` típus definícióját is.

Változó

Változó *A változó olyan programegység, amely a hozzá rendelt adattípus értékhalmazából műveletek hatására tetszőleges értéket felvehet, és értékét a program végrehajtása során akárhányszor megváltoztathatjuk.*

- A változókban tehát adott típusú értékeket tudunk tárolni későbbi felhasználás céljából.
- A változó végső soron a memória egy (az adattípusa által adott méretű) meghatározott része.
- A változó értéke kezdetben definiálatlan, és az marad, amíg valamilyen művelettel értéket nem adunk neki.

Változó

Deklaráció és definíció

- Változók létrehozása

```
típus változó-azonosító;
```

alakú deklarációval történik.

```
int magassag;
double forint, euro;
```

- A változók alapvetően értékadó művelettel kapnak értéket (definíció)

```
változó-azonosító = érték;
```

ahol az érték – ami bonyolult kifejezéssel is megadható – adattípusa megegyezik a változó adattípusával.

```
magassag = 100;
magassag = (magassag + 83);
```

Utasítások sorozata

- A C nyelvben az utasításokat a `;` zárja le, ez az, ami „utasítást csinál a kifejezésből”.
  - Ez tulajdonképpen azt jelenti a gép számára, hogy „és most számold ki amit eddig leírtam”.
  - Egy C függvény törzse a vezérlési szerkezetektől eltekintve nem más, mint kifejezések kiértékelésének sorozata. Ez olyannyira igaz, hogy például a

```
0;
```

    egy teljesen jó (bár pont annyira felesleges) utasítás.
  - Az egyes C műveleteknek lehetnek „mellékhatásai”, amik gyakran fontosabbak, mint maga a kifejezés eredménye (pl. `=` vagy `printf()`).
- Ha az utasításokat adott sorrendben kell végrehajtani, akkor egyszerűen `{` és `}` jelek között az adott sorrendben egymás után leírjuk őket.

Pénzváltás

penzvalto.c (v0.3) [1–15]

```c
/* Pénzváltás rögzített árfolyamon.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Augusztus 31. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

int main() {
    double forint;
    printf("Hány forintot váltsunk? ");
    scanf("%lf", &forint);
    /* TESZT */ printf("%lf\n", forint);
    return 0;
}
```

Definíció

Definíció *Egy programkomponens definíciója egy „érték” hozzárendelése a komponens azonosítójához.*

- Az „érték” a komponens deklarációjában meghatározott „típusú” kell legyen. (Művelet „értéke” pl. egy utasítássorozat.)
- A program egy adott pontján csak azoknak a komponenseknek az értékét szabad felhasználni, amelyeket e pontot megelőzően már definiáltunk, ellenkező esetben a program nem fordítható, nem szerkeszthető, vagy működése véletlenszerű, akár hibás is lehet.

Literál

Literál *A literál a program forráskódjában valamilyen típusú konstans érték közvetlen, vagyis nem külön azonosítóval ellátott megadása. Típusa az értékleírás által meghatározott adattípus.*

- A literál tehát nem más, mint valamely típus konkrét értékének programbéli leírása (pl. 3.141592654).

Műveletek számokkal

Matematikai kifejezések

- Az egész adattípus műveleteire teljesülnek az aritmetika ismert azonosságai, feltéve, hogy a művelet eredménye az adattípus értékhalmazába esik.
- Ha a művelet eredménye nem esne az adattípus értékhalmazába, túl- vagy alulcsordulásról beszélünk.

```
2 * (3 * 7 + 5 * 4) ==  4 * 2 * 5 + 3 * 2 * 7
     2147483647 + 1 == -2147483648
```

- Ha a művelet eredménye az adattípus értékhalmazába esik is, a valós adattípus műveleteire csak adott pontossággal teljesülnek az aritmetika ismert azonosságai.

```
abs( sin(M_PI_4) - cos(M_PI_4) ) <= 1e-13
     3.1415927e20 + 2.718281e-20 == 3.1415927e20
```

Műveletek számokkal

A / művelet

- A / jelölhet maradékos osztást de valós osztást is.
  - Ha a / művelet mindkét operandusa egész típusú, akkor a művelet egész osztást jelöl.
  - Ha a / művelet bármely operandusa valós típusú, akkor a művelet valós osztást jelöl.

```
15   / 6   == 2
15.0 / 6   == 2.5
15   / 6.0 == 2.5
15.0 / 6.0 == 2.5
```

Pénzváltás

penzvalto.c (v1.0) [1–16]

```c
/* Pénzváltás rögzített árfolyamon.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2022. Augusztus 30. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

int main() {
    double forint, euro;
    printf("Hány forintot váltsunk? ");
    scanf("%lf", &forint);
    euro = forint / 384.347;                                    /* 2023-09-14 */
    printf("%lf HUF = %lf EUR\n", forint, euro);
    return 0;
}
```

Tartalom

Pénzváltás

penzvalto.c (v2.0) [1–21]

```c
/* Pénzváltás rögzített árfolyamon.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2022. Augusztus 30. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

#define EUR 384.347                                    /* 2023-09-14 */

double valtas(double forint, double arfolyam) {
    return forint / arfolyam;
}

int main() {
    double osszeg;
    printf("Hány forintot váltsunk? ");
    scanf("%lf", &osszeg);
    printf("%lf HUF = %lf EUR\n", osszeg, valtas(osszeg, EUR));
    return 0;
}
```

Pénzváltás

A probléma

- A probléma nem változott.
- A feladat: oldjuk meg „szebben”.

Pénzváltás

penzvalto.c (v1.0) [1–16]

```c
/* Pénzváltás rögzített árfolyamon.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2022. Augusztus 30. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

int main() {
    double forint, euro;
    printf("Hány forintot váltsunk? ");
    scanf("%lf", &forint);
    euro = forint / 384.347;                                    /* 2023-09-14 */
    printf("%lf HUF = %lf EUR\n", forint, euro);
    return 0;
}
```

Egy C program felépítése

Teljesebb program

- Egy kevésbé egyszerű C program így néz ki:

```
/* A program adatai */

#include <stdio.h>

Konstansdefiníciók

Függvénydefiníciók

int main() {

    Változódeklarációk

    Utasítások

    return 0;
}
```

Konstans

Konstans *A konstans olyan komponense a programnak, amely a definíciójában megadott értéket azonosítja, és ez az érték a program végrehajtása során nem változtatható meg. Típusa a definíciója által meghatározott adattípus.*

- A konstans tehát egy konkrét érték elnevezéseként is felfogható (pl. π).

Konstans

Deklaráció és definíció

- Egy programot konstansok használata nélkül is meg lehet írni, de a program olvashatóságán, átláthatóságán, módosíthatóságán sokat javítanak.
- A konstansok deklarációja és definíciója C nyelven egybeesik, és a `#define` kulcsszó segítségével történik.

```
#define N   42
#define PI  3.1415926536
#define EPS 1e-10
```

- Ezek a C-ben valódi konstansok, úgy viselkednek, mintha a helyükre a tényleges értékeket (literálokat) írtuk volna (és valójában ez is történik).

Pénzváltás

penzvalto.c (v1.1) [1–18]

```c
/* Pénzváltás rögzített árfolyamon.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2022. Augusztus 30. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

#define EUR 384.347                                    /* 2023-09-14 */

int main() {
    double forint, euro;
    printf("Hány forintot váltsunk? ");
    scanf("%lf", &forint);
    euro = forint / EUR;
    printf("%lf HUF = %lf EUR\n", forint, euro);
    return 0;
}
```

Kifejezés

Kifejezés *Programkomponens, egy adattípus értékének műveleteket is tartalmazó jelölése.*

- A kifejezés által jelölt értéket a kifejezés kiértékelése határozza meg. Ez függ a műveletek
  - prioritásától és

```
 0<a   &&   a<b   &&   b<c   &&     a*a + b*b    ==   c*c
(0<a) && (a<b) && (b<c) && (((a*a)+(b*b)) == (c*c))
```

  - asszociativitásától.

```
 9 - 1   -    8 / 4 / 2   == 7
(9 - 1) - ((8 / 4) /2) == 7
```

Pénzváltás

penzvalto.c (v1.2) [1–17]

```c
/* Pénzváltás rögzített árfolyamon.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2022. Augusztus 30. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

#define EUR 384.347                                    /* 2023-09-14 */

int main() {
    double forint;
    printf("Hány forintot váltsunk? ");
    scanf("%lf", &forint);
    printf("%lf HUF = %lf EUR\n", forint, forint / EUR);
    return 0;
}
```

Függvény

Függvény *A függvény a matematikai értelemben vett függvény általánosítása (bővítése), gyakorlatilag egy (rész)algoritmus megvalósítása.*

- Egy-egy függvény valamilyen bemenő adatok alapján kiszámol egy értéket, mint ahogyan azt a matematikában már megszokhattuk.
- Vannak olyan függvények, amiknek valamilyen jól definiált mellékhatása is van a visszatérési érték kiszámítása mellett, például a szöveget megjelenítő függvények (`printf(...)`), adatbevitelt kezelő függvények (`scanf(...)`).
- Adott esetben az is előfordulhat, hogy egy függvénynek csak a mellékhatása fontos, és (matematikai értelemben) nem ad vissza semmilyen értéket. (Az ilyen függvényeket nevezhetjük eljárásnak.)

Függvény

Deklaráció és definíció [1/2]

- Egy C nyelven írt program tulajdonképpen nem más, mint függvények (megfelelően strukturált és rendezett) összessége.
- Függvényeket lehet deklarálni, definiálni és meghívni.
  - Deklarációnál csak azt mondjuk meg, hogy mennyi és milyen típusú paraméterekből milyen típusú értéket fog kiszámolni a függvényünk.
  - Definíciónál meg kell adnunk az algoritmust (is), hogy hogyan számoljon.
  - A függvényhívásnál pedig konkrét argumentumokra alkalmazzuk a függvényt, és a kiszámított értéket felhasználhatjuk további számolásainkhoz.
    - Természetesen egy függvénynek a híváskor pontosan annyi és olyan típusú argumentumot kell átadni, amennyi és amilyen paraméterrel deklarálva lett.

Függvény

Deklaráció és definíció [2/2]

- Függvény deklaráció

```
int f(int a, int b);
```

- Függvény definíció (egyben deklaráció is)

```
int f(int a, int b) {
    return a + b;
}
```

- Függvényhívás

```
int c;
c = f(3, 5);
```

A `return` utasítás

- Minden függvényben szerepelnie kell legalább egy `return` utasításnak.
- Ha a függvényben egy ilyen utasítást hajtunk végre, akkor a függvény értékének kiszámítása befejeződik. A hívás helyén a függvény a `return` által kiszámított értéket veszi fel.

```
int f(int a, int b) {
    return a + b;
}
```

A `main` függvény

- A C nyelvben a `main` függvénynek kitüntetett szerepe van.
- Amikor elindítjuk a programot, a vezérlés a `main` függvény elején indul, tehát ez a függvény viselkedik főprogramként.
  - Az operációs rendszertől ez a függvény is kaphat paramétereket, de ezekkel egyelőre nem foglalkozunk.
- A `main` által kiszámított értéket szintén az operációs rendszer értelmezi (miután befejeződött a program).

Pénzváltás

penzvalto.c (v2.0) [1–21]

```c
/* Pénzváltás rögzített árfolyamon.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2022. Augusztus 30. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

#define EUR 384.347                                    /* 2023-09-14 */

double valtas(double forint, double arfolyam) {
    return forint / arfolyam;
}

int main() {
    double osszeg;
    printf("Hány forintot váltsunk? ");
    scanf("%lf", &osszeg);
    printf("%lf HUF = %lf EUR\n", osszeg, valtas(osszeg, EUR));
    return 0;
}
```

Tartalom

Röppálya határértékeinek számítása

roppalya.c (v1.0) [1–28]

```c
/* Röppálya közelítő számítása.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Augusztus 31. Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

#define G 9.80665

int main() {
    double v0, vx, vy, alfa, t, sx, sy;
    printf("Kezdősebesség (m/s)? "); scanf("%lf", &v0);
    printf("Szög (fok)? "); scanf("%lf", &alfa);
    if (0.0 <= alfa && alfa <= 90.0 && 0.0 <= v0) {
        vy = v0 * sin(alfa / 90.0 * M_PI / 2.0);
        t  = vy / G;
        sy = G / 2.0 * t * t;
        printf("A röppálya legnagyobb magassága: %lf m\n", sy);
        vx = v0 * cos(alfa / 90.0 * M_PI / 2.0);
        sx = vx * 2.0 * t;
        printf("A lövedék távolsága földetéréskor: %lf m\n", sx);
    } else {
        printf("Hibás adatok!\n");
    }
    return 0;
}
```

Röppálya határértékeinek számítása

A probléma

- Eldobunk egy testet, milyen magasra megy, és hová esik?
- Csak egyszerűen: sík terepen, légellenállás nélkül, pontszerű testtel, közvetlenül a felszínről dobjuk.

Röppálya határértékeinek számítása

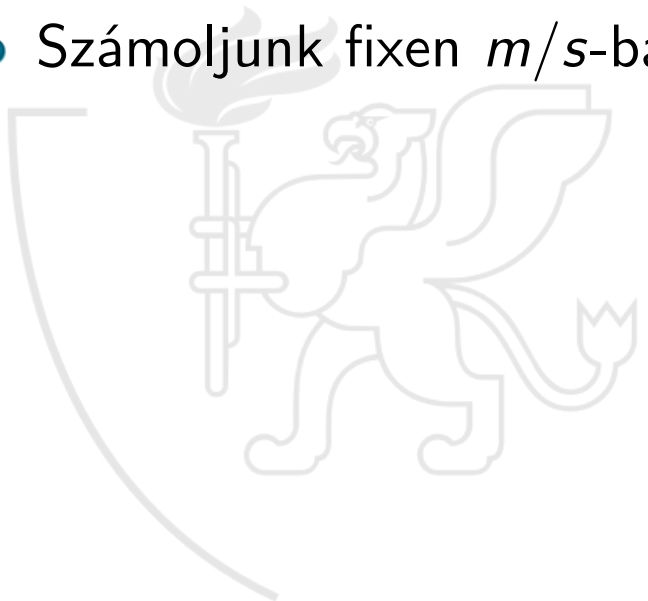
roppalya.c (v0.1) [1–9]

```c
/* Röppálya közelítő számítása.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Augusztus 31. Gergely Tamás, gertom.inf.u-szeged.hu
 */

int main() {
    return 0;
}
```

Röppálya határértékeinek számítása

Az elmélet

- Egy kis fizika
  - google search, ha nincs meg középiskolából `https://hu.wikipedia.org/wiki/Ferde_haj%C3%ADt%C3%A1s`
  - $v_x = v_0 \cdot \cos \alpha$
  - $v_y = v_0 \cdot \sin \alpha - g \cdot t$
  - $x = v_0 \cdot t \cdot \cos \alpha$
  - $y = v_0 \cdot t \cdot \sin \alpha - \frac{g}{2} \cdot t^2$
- Számoljunk fixen m/s-ban.



Matematikai függvények és konstansok

- A matematikai függvények, mint például a `sin`, `cos`, `log` vagy `exp` nem részei a nyelvnek, de egy standard C könyvtárban össze vannak gyűjtve.
- Használatukhoz az

```c
#include <math.h>
```

szükséges, illetve fordításkor a gcc-nek meg kell még adni a `-lm` kapcsolót is.

```c
double cos(double x);
double sin(double x);
double log(double x);
double exp(double x);
```

```c
cos(3.14159265)
sin(0.0)
log(2.718281)
exp(x)
```

Matematikai függvények és konstansok

- A matematikai konstansok, mint például a π vagy e szintén nem részei a nyelvnek, sőt, a C szabványok szerint (ANSI, C99, C11, C18) még a matematikai függvénykönyvtárban sem definiáltak.
- A legtöbb fordító ugyanakkor saját kiegészítésként definiál néhány hasznos konstanst. Ezek használatához szintén az

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

szükséges (de a `-lm` kapcsoló már nem).

```
#define M_E 2.7182818284590452354
#define M_PI 3.14159265358979323846
#define M_PI_2 1.57079632679489661923
#define M_SQRT2 1.41421356237309504880
#define M_SQRT1_2 0.70710678118654752440
```

```
log(M_E)
x / 180 * M_PI
cos(M_PI_2)
a * M_SQRT2
3.0 * M_SQRT1_2
```

Röppálya határértékeinek számítása

roppalya.c (v0.2) [1–19]

```c
/* Röppálya közelítő számítása.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Augusztus 31. Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

int main() {
    double v0, vx, vy, alfa;
    /* TESZT */ v0 = 10.0;
    /* TESZT */ alfa = 30.0;
    vy = v0 * sin(alfa / 90.0 * M_PI / 2.0);
    /* TESZT */ printf("Függőleges␣kezdősebesség␣(vy):␣%lf␣m/s\n", vy);
    vx = v0 * cos(alfa / 90.0 * M_PI / 2.0);
    /* TESZT */ printf("Vízszintes␣kezdősebesség␣(vx):␣%lf␣m/s\n", vx);
    return 0;
}
```

Röppálya határértékeinek számítása

Tesztelés

- Szükségünk lesz a gravitáció értékére, de a tesztelés kedvéért egyelőre válasszunk valami emészthető értéket.

Röppálya határértékeinek számítása

roppalya.c (v0.3) [1–23]

```c
/* Röppálya közelítő számítása.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Augusztus 31. Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

#define G 10.0

int main() {
    double v0, vx, vy, alfa, t;
    /* TESZT */ v0 = 10.0;
    /* TESZT */ alfa = 30.0;
    vy = v0 * sin(alfa / 90.0 * M_PI / 2.0);
    /* TESZT */ printf("Függőleges kezdősebesség (vy): %lf m/s\n", vy);
    t  = vy / G;
    /* TESZT */ printf("Repülési idő a csúcspontig: %lf s\n", t);
    vx = v0 * cos(alfa / 90.0 * M_PI / 2.0);
    /* TESZT */ printf("Vízszintes kezdősebesség (vx): %lf m/s\n", vx);
    return 0;
}
```

Röppálya határértékeinek számítása

Tesztelés

- És mi van akkor, ha lefelé hajítunk?

Röppálya határértékeinek számítása

roppalya.c (v0.4) [1–23]

```c
/* Röppálya közelítő számítása.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Augusztus 31. Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

#define G 10.0

int main() {
    double v0, vx, vy, alfa, t;
    /* TESZT */ v0 = 10.0;
    /* TESZT */ alfa = -30.0;
    vy = v0 * sin(alfa / 90.0 * M_PI / 2.0);
    /* TESZT */ printf("Függőleges kezdősebesség (vy): %lf m/s\n", vy);
    t  = vy / G;
    /* TESZT */ printf("Repülési idő a csúcspontig: %lf s\n", t);
    vx = v0 * cos(alfa / 90.0 * M_PI / 2.0);
    /* TESZT */ printf("Vízszintes kezdősebesség (vx): %lf m/s\n", vx);
    return 0;
}
```

Logikai értékek műveletei

Logikai műveletek

- `_Bool`→`_Bool`
  (*logikai*→*logikai*)
  egy operandusú műveletek
  - `!` tagadás, negáció
- `_Bool`×`_Bool`→`_Bool`
  (*logikai*×*logikai*→*logikai*)
  két operandusú műveletek
  - `&&` logikai *és*
  - `||` logikai *vagy*

```
!a
 a  &&  b
 a  ||  b
```

Logikai kifejezések

Rövidített logikai kiértékelés C-ben

- A logikai kifejezések kiértékelése mindig a rövidített kiértékelés szerint történik, vagyis
  - Az `A || B` kifejezés rövidített kiértékelése során először kiértékelődik az `A` logikai tag, ha ennek értéke igaz, akkor a `B` tag kiértékelése elmarad és természetesen a kifejezés értéke igaz (1) lesz.
  - Az `A && B` kifejezés rövidített kiértékelése során először kiértékelődik az `A` logikai tényező, ha ennek értéke hamis, akkor a `B` tényező kiértékelése elmarad és természetesen a kifejezés értéke hamis (0) lesz.

Az `if` utasítás

- Ha valamilyen feltétel alapján egyik vagy másik utasítást akarjuk végrehajtani.
  - Végre kell-e hajtani valamit?
    - Számoljuk ki `f` abszolút értékét!

```
if (f < 0)
  f = -f;
```

```
if (f < 0) {
  f = -f;
}
```

  - Két utasítás közül az egyiket hajtsuk végre!
    - `a` és `b` közül melyik a kisebb érték?

```
if (a <= b)
  k = a;
else
  k = b;
```

```
if (a <= b) {
  k = a;
} else {
  k = b;
}
```

Röppálya határértékeinek számítása

roppalya.c (v0.5) [1–27]

```c
/* Röppálya közelítő számítása.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Augusztus 31. Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

#define G 10.0

int main() {
    double v0, vx, vy, alfa, t;
    /* TESZT */ v0 = 10.0;
    /* TESZT */ alfa = -30.0;
    if (0.0 <= alfa && alfa <= 90.0 && 0.0 <= v0) {
        vy = v0 * sin(alfa / 90.0 * M_PI / 2.0);
        /* TESZT */ printf("Függőleges kezdősebesség (vy): %lf m/s\n", vy);
        t  = vy / G;
        /* TESZT */ printf("Repülési idő a csúcspontig: %lf s\n", t);
        vx = v0 * cos(alfa / 90.0 * M_PI / 2.0);
        /* TESZT */ printf("Vízszintes kezdősebesség (vx): %lf m/s\n", vx);
    } else {
        printf("Hibás adatok!\n");
    }
    return 0;
}
```

Röppálya határértékeinek számítása

Fejlesztés

- Számoljuk és írassuk ki a kívánt értékeket.

Röppálya határértékeinek számítása

roppalya.c (v0.6) [1–28]

```c
/* Röppálya közelítő számítása.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Augusztus 31. Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

#define G 10.0

int main() {
    double v0, vx, vy, alfa, t, sx, sy;
    /* TESZT */ v0 = 10.0;
    /* TESZT */ alfa = 45.0;
    if (0.0 <= alfa && alfa <= 90.0 && 0.0 <= v0) {
        vy = v0 * sin(alfa / 90.0 * M_PI / 2.0);
        t  = vy / G;
        sy = G / 2.0 * t * t;
        printf("A röppálya legnagyobb magassága: %lf m\n", sy);
        vx = v0 * cos(alfa / 90.0 * M_PI / 2.0);
        sx = vx * 2.0 * t;
        printf("A lövedék távolsága földetéréskor: %lf m\n", sx);
    } else {
        printf("Hibás adatok!\n");
    }
    return 0;
}
```

Röppálya határértékeinek számítása

Fejlesztés

- Végül, térjünk át felhasználói inputra, és használjunk valós gravitációs értéket (nem feledve, hogy m/s^2-ben kell megadni).

Röppálya határértékeinek számítása

roppalya.c (v1.0) [1–28]

```c
/* Röppálya közelítő számítása.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Augusztus 31. Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

#define G 9.80665

int main() {
    double v0, vx, vy, alfa, t, sx, sy;
    printf("Kezdősebesség (m/s)? "); scanf("%lf", &v0);
    printf("Szög (fok)? "); scanf("%lf", &alfa);
    if (0.0 <= alfa && alfa <= 90.0 && 0.0 <= v0) {
        vy = v0 * sin(alfa / 90.0 * M_PI / 2.0);
        t  = vy / G;
        sy = G / 2.0 * t * t;
        printf("A röppálya legnagyobb magassága: %lf m\n", sy);
        vx = v0 * cos(alfa / 90.0 * M_PI / 2.0);
        sx = vx * 2.0 * t;
        printf("A lövedék távolsága földetéréskor: %lf m\n", sx);
    } else {
        printf("Hibás adatok!\n");
    }
    return 0;
}
```

Tartalom

Röppálya szimuláció

roppalya-szim.c (v1.0) [1–15]

```c
/* Röppálya közelítő számítása.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Augusztus 31. Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

#define G 9.80665

typedef struct {
    double x;
    double y;
} vektor_t;
```

Röppálya szimuláció

roppalya-szim.c (v1.0) [17–34]

```c
int main() {
    double v0, alfa, dt, t;
    vektor_t v, s;
    s.x = s.y = t = 0.0;
    printf("Kezdősebesség (m/s)? "); scanf("%lf", &v0);
    printf("Szög (fok)? "); scanf("%lf", &alfa);
    printf("Delta t (s)? "); scanf("%lf", &dt);
    v.x = v0 * cos(alfa / 90.0 * M_PI / 2.0);
    v.y = v0 * sin(alfa / 90.0 * M_PI / 2.0);
    while (s.y >= 0.0) {
        printf("%lf sec: poz(%lf, %lf); seb[%lf, %lf]\n", t, s.x, s.y, v.x, v.y);
        s.x += v.x * dt;
        s.y += v.y * dt - G / 2.0 * dt * dt;
        v.y -= G * dt;
        t   += dt;
    }
    return 0;
}
```

Röppálya szimulációja

A probléma

- Eldobunk egy testet, milyen röppályán mozog?
- Csak egyszerűen: sík terepen, légellenállás nélkül, pontszerű testtel, közvetlenül a felszínről dobjuk.
- Egyenletes időközönként figyeljük, hogy hol van a test.
- Emlékszünk még?
  - $v_{0,x} = v_0 \cdot \cos \alpha$
  - $v_{0,y} = v_0 \cdot \sin \alpha$
  - $\Delta s_x = v_x \cdot \Delta t$
  - $\Delta s_y = v_y \cdot \Delta t \ (+\frac{g}{2} \cdot \Delta t^2)$
  - $\Delta v_x = 0$
  - $\Delta v_y = g \cdot \Delta t$
- Számoljunk fixen m/s-ban.

Röppálya szimuláció

roppalya-szim.c (v0.2) [1–18]

```c
/* Röppálya közelítő számítása.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Augusztus 31. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

int main() {
    double v0, alfa, dt;
    printf("Kezdősebesség (m/s)? "); scanf("%lf", &v0);
    printf("Szög (fok)? "); scanf("%lf", &alfa);
    printf("Delta t (s)? "); scanf("%lf", &dt);
    /* TESZT */ printf("Kezdősebesség %lf m/s.\n", v0);
    /* TESZT */ printf("Szög %lf fok.\n", alfa);
    /* TESZT */ printf("Delta t %lf s.\n", dt);
    return 0;
}
```

Típusdefiníció C-ben

- A C nyelvben lehetőségünk van típusok tetszésünk szerinti elnevezésére, azaz típusdefinícióra melyet a `typedef` kulcsszó vezet be, alakja:

```
typedef típus új_típusnév;
```

- A definíciótól kezdve a `típus`-ra az `új_típusnév` azonosítóval (is) hivatkozhatunk.
- A típusdefinícióval élhetünk például akkor, mikor több helyen kell ugyanolyan típusú változót deklarálni, de ez a típus
  - a jövőbeni fejlesztések során esetleg változhat, vagy
  - egy bonyolult módon megadható típus, amit nehézkes lenne többször leírni (és a többszöri leírás mellesleg hibaforrás is)!

```
typedef unsigned short int u16;
```

A struct típus

- Viszonylag gyakran szükség lehet arra, hogy (esetleg) különböző típusú összetartozó adatokat egységként kezeljünk.
- C-ben ennek megvalósítására a struct típusképzés használható.

```
typedef struct {
    short int sorszam;
    double    atlag;
    double    maximum;
    double    minimum;
} eredmeny;

eredmeny csoport;

csoport.sorszam = 42;
csoport.atlag   = 41.25;
csoport.maximum = 80.0;
csoport.minimum = 5.0;
```

Röppálya szimuláció

roppalya-szim.c (v0.3) [1–25]

```c
/* Röppálya közelítő számítása.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Augusztus 31. Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

typedef struct {
    double x;
    double y;
} vektor_t;

int main() {
    double v0, alfa, dt;
    vektor_t v;
    printf("Kezdősebesség (m/s)? "); scanf("%lf", &v0);
    printf("Szög (fok)? "); scanf("%lf", &alfa);
    printf("Delta t (s)? "); scanf("%lf", &dt);
    v.x = v0 * cos(alfa / 90.0 * M_PI / 2.0);
    v.y = v0 * sin(alfa / 90.0 * M_PI / 2.0);
    /* TESZT */ printf("vx: %lf; vy: %lf\n", v.x, v.y);
    return 0;
}
```

A C értékadó műveletei

- C-ben a

```
v = v ⊙ e
```

alakú értékadásokat

```
v ⊙= e
```

alakban rövidíthetjük, ahol a $\odot$ tetszőleges műveletet jelölhet.

```
x = 5; x += 2; x == 7;
x = 5; x -= 2; x == 3;
x = 5; x *= 2; x == 10;
x = 5; x /= 2; x == 2;
x = 5; x %= 2; x == 1;
```

- A teljes művelet mint kifejezés eredménye a kiszámolt új érték lesz.

A `while` utasítás

- Ha valamilyen műveletet mindaddig végre kell hajtani, amíg egy feltétel igaz a `while` utasítás (is) használható.
  - A művelet végrehajtása nem szükséges a feltétel kiértékeléséhez.
  - A feltétel ellenőrzése a művelet előtt történik, így ha a feltétel kezdetben hamis volt, a műveletet egyszer sem hajtjuk végre.
- Adjuk meg, hogy 0 fokról indulva adott mértékű pozitív irányú forgás után milyen irányban áll egy mutató.

```
while (360.0 <= szog) {
    szog -= 360.0;
}
```

A `do while` utasítás

- Ha valamilyen műveletet mindaddig végre kell hajtani, amíg egy feltétel igaz a `do while` utasítás (is) használható.
  - A művelet végrehajtása szükséges a feltétel kiértékeléséhez.
  - A feltétel ellenőrzése a művelet után történik, így ha a feltétel kezdetben hamis volt, a műveletet akkor is legalább egyszer végrehajtjuk.
- Kérjünk egy 0 és 999 közötti véletlen számot, de zárjuk ki a 100 és 200 közötti számokat.

```
do {
    x = random() % 1000;
} while ((100 <= x) && (x <= 200));
```

Röppálya szimuláció

roppalya-szim.c (v0.4) [1–15]

```c
/* Röppálya közelítő számítása.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Augusztus 31. Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

#define G 9.80665

typedef struct {
    double x;
    double y;
} vektor_t;
```

Röppálya szimuláció

roppalya-szim.c (v0.4) [17–33]

```
17 int main() {
18     double v0, alfa, dt, t, tmax;
19     vektor_t v;
20     t = 0.0;
21     printf("Kezdősebesség␣(m/s)?␣"); scanf("%lf", &v0);
22     printf("Szög␣(fok)?␣"); scanf("%lf", &alfa);
23     printf("Delta␣t␣(s)?␣"); scanf("%lf", &dt);
24     v.x = v0 * cos(alfa / 90.0 * M_PI / 2.0);
25     v.y = v0 * sin(alfa / 90.0 * M_PI / 2.0);
26     tmax = 2.0 * v.y / G;
27     while (t <= tmax) {
28         printf("%lf␣sec:␣seb[%lf,␣%lf]\n", t, v.x, v.y);
29         v.y -= G * dt;
30         t   += dt;
31     }
32     return 0;
33 }
```

Értékadó művelet

- Az értékadás jele az =, de ez művelet, és nem utasítás.
- Vagyis a

```
változóazonosító = kifejezés
```

  művelet eredménye a `kifejezés` aktuális értéke, amit a művelet „mellékhatásaként” a megfelelő programkomponensben is eltárolunk.
- Természetesen nincs akadálya a művelet többszöri alkalmazásának.
- Az = művelet jobb-asszociatív és alacsony prioritású.

```
i =   j =   k = 1;
i = (j = (k = 1));
```

Röppálya szimuláció

roppalya-szim.c (v0.5) [17–35]

```c
int main() {
    double v0, alfa, dt, t, tmax;
    vektor_t v, s;
    s.x = s.y = t = 0.0;
    printf("Kezdősebesség (m/s)? "); scanf("%lf", &v0);
    printf("Szög (fok)? "); scanf("%lf", &alfa);
    printf("Delta t (s)? "); scanf("%lf", &dt);
    v.x = v0 * cos(alfa / 90.0 * M_PI / 2.0);
    v.y = v0 * sin(alfa / 90.0 * M_PI / 2.0);
    tmax = 2.0 * v.y / G;
    while (t <= tmax) {
        printf("%lf sec: poz(%lf, %lf); seb[%lf, %lf]\n", t, s.x, s.y, v.x, v.y);
        s.x += v.x * dt;
        s.y += v.y * dt;
        v.y -= G * dt;
        t   += dt;
    }
    return 0;
}
```

Röppálya szimulációja

Fejlesztés

- Felesleges előre kiszámolni a repülési időt, elég, ha a tárgy aktuális magasságát figyeljük.

Röppálya szimuláció

roppalya-szim.c (v0.6) [17–34]

```
17 int main() {
18     double v0, alfa, dt, t;
19     vektor_t v, s;
20     s.x = s.y = t = 0.0;
21     printf("Kezdősebesség␣(m/s)?␣"); scanf("%lf", &v0);
22     printf("Szög␣(fok)?␣"); scanf("%lf", &alfa);
23     printf("Delta␣t␣(s)?␣"); scanf("%lf", &dt);
24     v.x = v0 * cos(alfa / 90.0 * M_PI / 2.0);
25     v.y = v0 * sin(alfa / 90.0 * M_PI / 2.0);
26     while (s.y >= 0.0) {
27         printf("%lf␣sec:␣poz(%lf,␣%lf);␣seb[%lf,␣%lf]\n", t, s.x, s.y, v.x, v.y);
28         s.x += v.x * dt;
29         s.y += v.y * dt;
30         v.y -= G * dt;
31         t   += dt;
32     }
33     return 0;
34 }
```

Röppálya szimulációja

Fejlesztés

- Tegyük kicsit pontosabbá a számítást.

Röppálya szimuláció

roppalya-szim.c (v1.0) [1–15]

```c
/* Röppálya közelítő számítása.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Augusztus 31. Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

#define G 9.80665

typedef struct {
    double x;
    double y;
} vektor_t;
```

Röppálya szimuláció

roppalya-szim.c (v1.0) [17–34]

```
17 int main() {
18     double v0, alfa, dt, t;
19     vektor_t v, s;
20     s.x = s.y = t = 0.0;
21     printf("Kezdősebesség␣(m/s)?␣"); scanf("%lf", &v0);
22     printf("Szög␣(fok)?␣"); scanf("%lf", &alfa);
23     printf("Delta␣t␣(s)?␣"); scanf("%lf", &dt);
24     v.x = v0 * cos(alfa / 90.0 * M_PI / 2.0);
25     v.y = v0 * sin(alfa / 90.0 * M_PI / 2.0);
26     while (s.y >= 0.0) {
27         printf("%lf␣sec:␣poz(%lf,␣%lf);␣seb[%lf,␣%lf]\n", t, s.x, s.y, v.x, v.y);
28         s.x += v.x * dt;
29         s.y += v.y * dt - G / 2.0 * dt * dt;
30         v.y -= G * dt;
31         t   += dt;
32     }
33     return 0;
34 }
```

Tartalom

Egy dátum az év hányadik napja?

evnapja.c (v1.0) [1–13]

```c
/* Egy YYYY-MM-DD formátumban megadott dátum az év hányadik napja?
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

typedef struct {
    int ev;
    int ho;
    int nap;
} datum_t;
```

Egy dátum az év hányadik napja?

evnapja.c (v1.0) [15–43]

```
15 int main() {
16     datum_t d;
17     int ev_napja;
18
19     scanf("%d-%d-%d", &d.ev, &d.ho, &d.nap);
20     ev_napja = d.nap;
21     switch (d.ho) {
22         case 12: ev_napja += 30;
23         case 11: ev_napja += 31;
24         case 10: ev_napja += 30;
25         case  9: ev_napja += 31;
26         case  8: ev_napja += 31;
27         case  7: ev_napja += 30;
28         case  6: ev_napja += 31;
29         case  5: ev_napja += 30;
30         case  4: ev_napja += 31;
31         case  3: ev_napja += (d.ev % 4) ?
32                              28 : ((d.ev % 100) ?
33                                    29 : ((d.ev % 400) ?
34                                          28 : 29
35                                    )
36                              );
37         case  2: ev_napja += 31;
38         case  1: break;
39         default: return 1;
40     }
41     printf("%d-%d-%d az év %d. napja\n", d.ev, d.ho, d.nap, ev_napja);
42     return 0;
43 }
```

Egy dátum az év hányadik napja?

A probléma

- Adott egy dátum `YYYY-MM-DD` alakban.
- Az adott évben ez hányadik nap a Gergely-naptár szerint?
- A dátum helyességével most ne foglalkozzunk.

Egy dátum az év hányadik napja?

evnapja.c (v0.2) [1–21]

```c
/* Egy YYYY-MM-DD formátumban megadott dátum az év hányadik napja?
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

typedef struct {
    int ev;
    int ho;
    int nap;
} datum_t;

int main() {
    datum_t d;

    scanf("%d-%d-%d", &d.ev, &d.ho, &d.nap);
    /* TESZT */ printf("%d-%d-%d\n", d.ev, d.ho, d.nap);
    return 0;
}
```

Egy dátum az év hányadik napja?

Fejlesztés

- Januárban egyszerűen a nap lesz a sorszám.

Egy dátum az év hányadik napja?

evnapja.c (v0.3) [1–23]

```c
/* Egy YYYY-MM-DD formátumban megadott dátum az év hányadik napja?
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

typedef struct {
    int ev;
    int ho;
    int nap;
} datum_t;

int main() {
    datum_t d;
    int ev_napja;

    scanf("%d-%d-%d", &d.ev, &d.ho, &d.nap);
    ev_napja = d.nap;
    printf("%d-%d-%d az év %d. napja\n", d.ev, d.ho, d.nap, ev_napja);
    return 0;
}
```

Egy dátum az év hányadik napja?

Fejlesztés

- Valójában, a sorszámhoz a nap értékét mindig hozzá kell adni, akármilyen hónap is van.
- Ha nem január van, akkor a január mind a 31 napja az adott nap előtt van.

Egy dátum az év hányadik napja?

evnapja.c (v0.4) [1–26]

```c
/* Egy YYYY-MM-DD formátumban megadott dátum az év hányadik napja?
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

typedef struct {
    int ev;
    int ho;
    int nap;
} datum_t;

int main() {
    datum_t d;
    int ev_napja;

    scanf("%d-%d-%d", &d.ev, &d.ho, &d.nap);
    ev_napja = d.nap;
    if (d.ho > 1) {
        ev_napja += 31;
    }
    printf("%d-%d-%d az év %d. napja\n", d.ev, d.ho, d.nap, ev_napja);
    return 0;
}
```

Egy dátum az év hányadik napja?

Fejlesztés

- Ha a januáron túl vagyunk, akkor a január mind a 31 napja az adott nap előtt van.
- Ha a februáron túl vagyunk, akkor a február mind a 28 napja az adott nap előtt van.
- Ha a márciuson túl vagyunk, akkor a március mind a 31 napja az adott nap előtt van.

Egy dátum az év hányadik napja?

evnapja.c (v0.6) [15–32]

```c
int main() {
    datum_t d;
    int ev_napja;

    scanf("%d-%d-%d", &d.ev, &d.ho, &d.nap);
    ev_napja = d.nap;
    if (d.ho > 1) {
        ev_napja += 31;
    }
    if (d.ho > 2) {
        ev_napja += 28;
    }
    if (d.ho > 3) {
        ev_napja += 31;
    }
    printf("%d-%d-%d␣az␣év␣%d.␣napja\n", d.ev, d.ho, d.nap, ev_napja);
    return 0;
}
```

Egy dátum az év hányadik napja?

Fejlesztés

- Vegyük észre, hogy ha nem vagyunk túl a januáron, akkor a többi hónapot teljesen felesleges ellenőrizni.

Egy dátum az év hányadik napja?

evnapja.c (v0.7) [15–32]

```c
int main() {
    datum_t d;
    int ev_napja;

    scanf("%d-%d-%d", &d.ev, &d.ho, &d.nap);
    ev_napja = d.nap;
    if (d.ho > 1) {
        ev_napja += 31;
        if (d.ho > 2) {
            ev_napja += 28;
            if (d.ho > 3) {
                ev_napja += 31;
            }
        }
    }
    printf("%d-%d-%d az év %d. napja\n", d.ev, d.ho, d.nap, ev_napja);
    return 0;
}
```

Egy dátum az év hányadik napja?

Fejlesztés

- A baj csak az, hogy a 11 hónap ellenőrzésével a kódunk . . .
  . . . nos, nem lesz szép.

Egy dátum az év hányadik napja?

evnapja.c (v0.8) [15–42]

```
15 int main() {
16     datum_t d;
17     int ev_napja;
18 
19     scanf("%d-%d-%d", &d.ev, &d.ho, &d.nap);
20     ev_napja = d.nap;
21     if (d.ho > 1) {
22         ev_napja += 31;
23         if (d.ho > 2) {
24             ev_napja += 28;
25             if (d.ho > 3) {
26                 ev_napja += 31;
27                 if (d.ho > 4) {
28                     ev_napja += 30;
29                     if (d.ho > 5) {
30                         ev_napja += 31;
31                         if (d.ho > 6) {
32                             ev_napja += 30;
33                             if (d.ho > 7) {
34                                 ev_napja += 31;
35                                 if (d.ho > 8) {
36                                     ev_napja += 31;
37                                     if (d.ho > 9) {
38                                         ev_napja += 30;
39                                         if (d.ho > 10) {
40                                             ev_napja += 31;
41                                             if (d.ho > 11) {
42                                                 ev_napja += 30;
```

Egy dátum az év hányadik napja?

evnapja.c (v0.8) [29–56]

```
29                      if (d.ho > 5) {
30                          ev_napja += 31;
31                          if (d.ho > 6) {
32                              ev_napja += 30;
33                              if (d.ho > 7) {
34                                  ev_napja += 31;
35                                  if (d.ho > 8) {
36                                      ev_napja += 31;
37                                      if (d.ho > 9) {
38                                          ev_napja += 30;
39                                          if (d.ho > 10) {
40                                              ev_napja += 31;
41                                              if (d.ho > 11) {
42                                                  ev_napja += 30;
43                                              }
44                                          }
45                                      }
46                                  }
47                              }
48                          }
49                      }
50                  }
51              }
52          }
53      }
54      printf("%d-%d-%d az év %d. napja\n", d.ev, d.ho, d.nap, ev_napja);
55      return 0;
56  }
```

A `switch` utasítás

- Ha egy kifejezés értéke alapján többféle utasítás közül kell választanunk, a `switch` utasítást használhatjuk.
  - Megadhatjuk, hogy hol kezdődjön és meddig tartson az utasítás-sorozat végrehajtása.
- Gyakorlati jegyből csináljunk háromfokozatú minősítést!

```
switch (gy) {
case 5:
case 4:  printf("jól␣");
case 3:
case 2:  printf("megfelelt");
         break;
case 1:  printf("nem␣felelt␣meg");
         break;
default: printf("nincs␣ilyen␣jegy");
         break;
}
```

Egy dátum az év hányadik napja?

evnapja.c (v0.9) [15–37]

```c
int main() {
    datum_t d;
    int ev_napja;

    scanf("%d-%d-%d", &d.ev, &d.ho, &d.nap);
    ev_napja = d.nap;
    switch (d.ho) {
        case 12: ev_napja += 30;
        case 11: ev_napja += 31;
        case 10: ev_napja += 30;
        case  9: ev_napja += 31;
        case  8: ev_napja += 31;
        case  7: ev_napja += 30;
        case  6: ev_napja += 31;
        case  5: ev_napja += 30;
        case  4: ev_napja += 31;
        case  3: ev_napja += 28;
        case  2: ev_napja += 31;
        case  1: break;
    }
    printf("%d-%d-%d␣az␣év␣%d.␣napja\n", d.ev, d.ho, d.nap, ev_napja);
    return 0;
}
```

Feltételes kifejezés

- A feltételes operátor a C nyelv egyetlen háromoperandusú művelete. A K&R könyv feltételes kifejezésnek említi.

```
kifejezés1 ? kifejezés2 : kifejezés3
```

- Először a `kifejezés`$_1$ kerül kiértékelésre, ha ez
  - igaz (nem 0), a kifejezés értéke `kifejezés`$_2$ lesz,
  - hamis (0), a kifejezés értéke `kifejezés`$_3$ lesz.

Egy dátum az év hányadik napja?

evnapja.c (v1.0) [1–13]

```c
/* Egy YYYY-MM-DD formátumban megadott dátum az év hányadik napja?
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

typedef struct {
    int ev;
    int ho;
    int nap;
} datum_t;
```

Egy dátum az év hányadik napja?

evnapja.c (v1.0) [15–43]

```
15 int main() {
16     datum_t d;
17     int ev_napja;
18
19     scanf("%d-%d-%d", &d.ev, &d.ho, &d.nap);
20     ev_napja = d.nap;
21     switch (d.ho) {
22         case 12: ev_napja += 30;
23         case 11: ev_napja += 31;
24         case 10: ev_napja += 30;
25         case  9: ev_napja += 31;
26         case  8: ev_napja += 31;
27         case  7: ev_napja += 30;
28         case  6: ev_napja += 31;
29         case  5: ev_napja += 30;
30         case  4: ev_napja += 31;
31         case  3: ev_napja += (d.ev % 4) ?
32                              28 : ((d.ev % 100) ?
33                                    29 : ((d.ev % 400) ?
34                                          28 : 29
35                                    )
36                              );
37         case  2: ev_napja += 31;
38         case  1: break;
39         default: return 1;
40     }
41     printf("%d-%d-%d az év %d. napja\n", d.ev, d.ho, d.nap, ev_napja);
42     return 0;
43 }
```

Tartalom

Csúszóátlag számítása adott elemszámra

csuszoatlag.c (v1.0) [1–26]

```c
/* Csúszóátlag számítása fix méretű tömbön.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

#define N 15

double atlag3(double v1, double v2, double v3) {
    return (v1 + v2 + v3) / 3;
}

int main() {
    double ertektomb[N], atlagtomb[N - 2];
    for (int i = 0; i < N; ++i) {
        printf("? "); scanf("%lf", &(ertektomb[i]));
    }
    for (int i = 0; i < N - 2; ++i)
        atlagtomb[i] = atlag3(ertektomb[i], ertektomb[i + 1], ertektomb[i + 2]);
    for (int i = 0; i < N - 2; ++i)
        printf("%lf;", atlagtomb[i]);
    putchar('\n');
    return 0;
}
```

Csúszóátlag számítása adott elemszámra

A probléma

- Adott egy 15 elemű számsorozat, két hét új COVID-19 fertőzöttjeinek napi száma.
- Számoljunk 3 napos átlagokat, hogy csillapítsuk egy-egy nap kiugró értékeit.

Tömb típus C-ben

- Algoritmusok tervezésekor gyakran előfordul, hogy azonos típusú adatok sorozatával kell dolgozni, vagy mert az input adatok sorozatot alkotnak, vagy mert a feladat megoldásához kell.
- C-ben ennek megvalósítására a tömb típusképzés használható.

```
char                 ct[5];
unsigned short int   it[20];
int                  matrix[10][10];

ct [0]  = 'A';
it [19] = 42;
printf("%ld", matrix[i][j]);
```

Tömb típus C-ben

Tömbindexelés

- C nyelvben a tömb indexelése minden esetben 0-val kezdődik, azaz egy

```
int t[20];
```

deklaráció esetén a tömb elemei `t[0]`, `t[1]`, ..., `t[19]` lesznek.

- Nincs viszont indexhatár-ellenőrzés, azaz a fenti deklaráció esetén például a `t[20]` vagy `t[-1]` elemekre is lehet hivatkozni, ez azonban nagyon csúnya futási hibákat eredményezhet. Például

```
double m[10][10];
```

deklaráció esetén `m[0][10]` ugyanaz mint `m[1][0]`.

Csúszóátlag számítása adott elemszámra

csuszoatlag.c (v0.1) [1–12]

```c
/* Csúszóátlag számítása fix méretű tömbön.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#define N 15

int main() {
    double ertektomb[N], atlagtomb[N - 2];
    return 0;
}
```

A `for` utasítás [1/2]

- Ha valamilyen műveletet sorban több értékére is végre kell hajtani, akkor a `for` utasítás (is) használható.
- C-ben a `for` utasítás általános alakja így néz ki:

```
for (kif1; kif2; kif3) utasítás;
```

ami egyenértékű ezzel:

```
kif1;
while (kif2) {
    utasítás;
    kif3;
}
```

A `for` utasítás [2/2]

- Hajtsunk végre egy műveletet adott számú alkalommal.
  - „És most büntetésből százszor leírod a nevedet!”

```
for (i = 0; i < 100; ++i) {
    printf("neved\n");
}
```

- Egy változót lépésenként ugyanúgy változtatunk.
  - Írassuk ki egy mértani sorozat elemeit a kezdőelem duplájáig.

```
for (x = a0; x < 2.0 * a0; x *= q) {
    printf("%lf\n", x);
}
```

A `break` és `continue` utasítások

- A C nyelvben a ciklusmag folyamatos végrehajtásának megszakítására két utasítás használható:

  `break` Megszakítja a ciklust, a program végrehajtása a ciklusmag utáni első utasítással folytatódik. Használható a `switch` utasításban is, hatására a program végrehajtása a `switch` utáni első utasítással folytatódik.

  `continue` Megszakítja a ciklusmag aktuális lefutását, a vezérlés a ciklus feltételének kiértékelésével (`while`, `do while`) illetve az inkrementáló kifejezés kiértékelésével (`for`) folytatódik.

Inkrementáló és dekrementáló műveletek

- A C nyelv tartalmaz két operátort, amelyekkel változók értékét lehet eggyel növelni vagy csökkenteni.

  `++` eggyel növeli a változó értékét

  `--` eggyel csökkenti a változó értékét

- Mindkettő használható *prefix* és *postfix* operátorként is.

```
i = 5; x = i++; x == 5; i == 6;
i = 5; x = ++i; x == 6; i == 6;
i = 5; x = i--; x == 5; i == 4;
i = 5; x = --i; x == 4; i == 4;
```

Csúszóátlag számítása adott elemszámra

csuszoatlag.c (v0.2) [1–20]

```c
/* Csúszóátlag számítása fix méretű tömbön.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

#define N 15

int main() {
    double ertektomb[N], atlagtomb[N - 2];
    for (int i = 0; i < N; ++i) {
        printf("? "); scanf("%lf", &(ertektomb[i]));
    }
    /* TESZT */ for (int i = 0; i < N; ++i)
    /* TESZT */     printf("%lf;", ertektomb[i]);
    /* TESZT */ printf("\n");
    return 0;
}
```

Csúszóátlag számítása adott elemszámra

Fejlesztés

- Készítsük el a főprogramot.
- Az átlagszámítás részleteivel egyelőre ne foglalkozzunk.

Csúszóátlag számítása adott elemszámra

csuszoatlag.c (v0.3) [1–26]

```c
/* Csúszóátlag számítása fix méretű tömbön.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

#define N 15

double atlag3(double v1, double v2, double v3) {
    /* TESZT */ return 0.0;
}

int main() {
    double ertektomb[N], atlagtomb[N - 2];
    for (int i = 0; i < N; ++i) {
        printf("? "); scanf("%lf", &(ertektomb[i]));
    }
    for (int i = 0; i < N - 2; ++i)
        atlagtomb[i] = atlag3(ertektomb[i], ertektomb[i + 1], ertektomb[i + 2]);
    for (int i = 0; i < N - 2; ++i)
        printf("%lf;", atlagtomb[i]);
    printf("\n");
    return 0;
}
```

Karakter adattípus a C nyelvben

- A `char` adattípus a C nyelv eleve definiált elemi adattípusa, értékkészlete 256 elemet tartalmaz.
- A `char` adattípus egészként is használható, de kimeneti eszközön karakterként jelenik meg.
  - Hogy melyik értékhez melyik karakter tartozik, az az alkalmazott kódtáblázattól függ.
  - Bizonyos karakterek (általában a rendezés szerint első néhány) vezérlő karakternek számítanak, és nem megjeleníthetők.
- A karaktereket aposztrófok (') között kell megadni.
  - A speciális karaktereket, illetve magát az aposztrófot (és végső soron tetszőleges karaktert is) escape-szekvenciákkal lehet megadni. Az escape-szekvenciákat a \ (backslash) karakterrel kell kezdeni.

Karakter adattípus a C nyelven

Példa

- Konvertáljunk egy tetszőleges számjegy karaktert (ch) a neki megfelelő egész számmá.

```
ch - '0'
```

- Konvertáljunk egy tetszőleges egyjegyű egészet (i) a neki megfelelő karakterré.

```
i + '0'
```

- Konvertáljunk kisbetűt nagybetűvé.

```
ch - 'a' + 'A'
```

- Konvertáljunk nagybetűt kisbetűvé.

```
ch - 'A' + 'a'
```

Csúszóátlag számítása adott elemszámra

csuszoatlag.c (v1.0) [1–26]

```c
/* Csúszóátlag számítása fix méretű tömbön.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

#define N 15

double atlag3(double v1, double v2, double v3) {
    return (v1 + v2 + v3) / 3;
}

int main() {
    double ertektomb[N], atlagtomb[N - 2];
    for (int i = 0; i < N; ++i) {
        printf("? "); scanf("%lf", &(ertektomb[i]));
    }
    for (int i = 0; i < N - 2; ++i)
        atlagtomb[i] = atlag3(ertektomb[i], ertektomb[i + 1], ertektomb[i + 2]);
    for (int i = 0; i < N - 2; ++i)
        printf("%lf;", atlagtomb[i]);
    putchar('\n');
    return 0;
}
```

Tartalom

Csúszóátlag számítás parancssorból

csuszoatlag.c (v2.0) [1–29]

```c
/* Csúszóátlag számítása.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

double atlag3(double v1, double v2, double v3) {
    return (v1 + v2 + v3) / 3;
}

int main(int argc, char *argv[]) {
    double ertektomb[3];
    if (argc < 4) {
        return 1;
    }
    ertektomb[0] = atof(argv[1]);
    ertektomb[1] = atof(argv[2]);
    for (int i = 3; i < argc; ++i) {
        ertektomb[2] = atof(argv[i]);
        printf("%lf;", atlag3(ertektomb[0], ertektomb[1], ertektomb[2]));
        ertektomb[0] = ertektomb[1];
        ertektomb[1] = ertektomb[2];
    }
    putchar('\n');
    return 0;
}
```

Csúszóátlag számítás parancssorból

A probléma

- Ugyanaz mint az előbb, de nem ismert előre a napok száma.
- A programunk a parancssorban kapott paraméterekből dolgozzon.

Csúszóátlag számítás parancssorból

csuszoatlag.c (v1.0) [1–26]

```c
/* Csúszóátlag számítása fix méretű tömbön.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

#define N 15

double atlag3(double v1, double v2, double v3) {
    return (v1 + v2 + v3) / 3;
}

int main() {
    double ertektomb[N], atlagtomb[N - 2];
    for (int i = 0; i < N; ++i) {
        printf("? "); scanf("%lf", &(ertektomb[i]));
    }
    for (int i = 0; i < N - 2; ++i)
        atlagtomb[i] = atlag3(ertektomb[i], ertektomb[i + 1], ertektomb[i + 2]);
    for (int i = 0; i < N - 2; ++i)
        printf("%lf;", atlagtomb[i]);
    putchar('\n');
    return 0;
}
```

Parancssori argumentumok kezelése

- A C nyelvet támogató környezetekben lehetőség van arra, hogy a végrehajtás megkezdésekor a programnak parancssori argumentumokat vagy paramétereket adjunk át.
- Amikor az operációs rendszer elindítja a programot, azaz meghívja a `main` függvényt, a hívásban két argumentum szerepelhet:
  - Az első (általában `argc`) azoknak a parancssori argumentumoknak a darabszáma, amelyekkel a programot meghívtuk.
  - A második argumentum (általában `argv`) egy mutató egy sztring-tömbre, amely a parancssori argumentumokat tartalmazza. Egy karakterlánc egy argumentumnak felel meg.
    - Megállapodás szerint `argv[0]` az a név, amellyel a programot hívták, így az `argc` értéke legalább 1.
    - Számíthatunk arra is, hogy `argv[argc]==NULL`.
- Mivel az `argv` egy mutatótömböt megcímző mutató, a `main` függvényt többféleképpen deklarálhatjuk.

Csúszóátlag számítás parancssorból

csuszoatlag.c (v1.1) [1–27]

```c
/* Csúszóátlag számítása.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

#define N 15

double atlag3(double v1, double v2, double v3) {
    return (v1 + v2 + v3) / 3;
}

int main(int argc, char *argv[]) {
    double ertektomb[N], atlagtomb[N - 2];
    for (int i = 0; i < argc - 1; ++i) {
        ertektomb[i] = atof(argv[i + 1]);
    }
    for (int i = 0; i < argc - 3; ++i)
        atlagtomb[i] = atlag3(ertektomb[i], ertektomb[i + 1], ertektomb[i + 2]);
    for (int i = 0; i < argc - 3; ++i)
        printf("%lf;", atlagtomb[i]);
    putchar('\n');
    return 0;
}
```

Csúszóátlag számítás parancssorból

Fejlesztés

- Egyszerűsítsünk: az utolsó két `for` ciklus összefésülhető egyetlen ciklussá.

Csúszóátlag számítás parancssorból

csuszoatlag.c (v1.2) [1–27]

```c
/* Csúszóátlag számítása.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

#define N 15

double atlag3(double v1, double v2, double v3) {
    return (v1 + v2 + v3) / 3;
}

int main(int argc, char *argv[]) {
    double ertektomb[N], atlagtomb[N - 2];
    for (int i = 0; i < argc - 1; ++i) {
        ertektomb[i] = atof(argv[i + 1]);
    }
    for (int i = 0; i < argc - 3; ++i) {
        atlagtomb[i] = atlag3(ertektomb[i], ertektomb[i + 1], ertektomb[i + 2]);
        printf("%lf;", atlagtomb[i]);
    }
    putchar('\n');
    return 0;
}
```

Csúszóátlag számítás parancssorból

Fejlesztés

- Egyszerűsítsünk: ha az átlagértékeket nem tároljuk, hanem egyből kiírjuk, akkor megszabadulhatunk a második tömbtől.

Csúszóátlag számítás parancssorból

csuszoatlag.c (v1.3) [1–26]

```c
/* Csúszóátlag számítása.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

#define N 15

double atlag3(double v1, double v2, double v3) {
    return (v1 + v2 + v3) / 3;
}

int main(int argc, char *argv[]) {
    double ertektomb[N];
    for (int i = 0; i < argc - 1; ++i) {
        ertektomb[i] = atof(argv[i + 1]);
    }
    for (int i = 0; i < argc - 3; ++i) {
        printf("%lf;", atlag3(ertektomb[i], ertektomb[i + 1], ertektomb[i + 2]));
    }
    putchar('\n');
    return 0;
}
```

Csúszóátlag számítás parancssorból

Fejlesztés

- Egy kis átrendezéssel a beolvasás egy részét is átvihetjük a második ciklusba.
- Így az első ciklus legfeljebb két lépésből áll.

Csúszóátlag számítás parancssorból

csuszoatlag.c (v1.4) [1–27]

```c
/* Csúszóátlag számítása.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

#define N 15

double atlag3(double v1, double v2, double v3) {
    return (v1 + v2 + v3) / 3;
}

int main(int argc, char *argv[]) {
    double ertektomb[N];
    for (int i = 0; i < argc - 1 && i < 2; ++i) {
        ertektomb[i] = atof(argv[i + 1]);
    }
    for (int i = 0; i < argc - 3; ++i) {
        ertektomb[i + 2] = atof(argv[i + 3]);
        printf("%lf;", atlag3(ertektomb[i], ertektomb[i + 1], ertektomb[i + 2]));
    }
    putchar('\n');
    return 0;
}
```

Csúszóátlag számítás parancssorból

Fejlesztés

- Fejtsük ki az első, kétlépéses ciklust.

Csúszóátlag számítás parancssorból

csuszoatlag.c (v1.5) [1–26]

```c
/* Csúszóátlag számítása.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

#define N 15

double atlag3(double v1, double v2, double v3) {
    return (v1 + v2 + v3) / 3;
}

int main(int argc, char *argv[]) {
    double ertektomb[N];
    ertektomb[0] = atof(argv[1]);
    ertektomb[1] = atof(argv[2]);
    for (int i = 0; i < argc - 3; ++i) {
        ertektomb[i + 2] = atof(argv[i + 3]);
        printf("%lf;", atlag3(ertektomb[i], ertektomb[i + 1], ertektomb[i + 2]));
    }
    putchar('\n');
    return 0;
}
```

Csúszóátlag számítás parancssorból

Fejlesztés

- Egy kis korrekció: ha túl kevés a paraméter, akkor lényegi dolgot amúgy sem csinálnánk, de a ciklus kifejtése erre nem volt tekintettel.

Csúszóátlag számítás parancssorból

csuszoatlag.c (v1.6) [1–29]

```c
/* Csúszóátlag számítása.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

#define N 15

double atlag3(double v1, double v2, double v3) {
    return (v1 + v2 + v3) / 3;
}

int main(int argc, char *argv[]) {
    double ertektomb[N];
    if (argc < 4) {
        return 1;
    }
    ertektomb[0] = atof(argv[1]);
    ertektomb[1] = atof(argv[2]);
    for (int i = 0; i < argc - 3; ++i) {
        ertektomb[i + 2] = atof(argv[i + 3]);
        printf("%lf;", atlag3(ertektomb[i], ertektomb[i + 1], ertektomb[i + 2]));
    }
    putchar('\n');
    return 0;
}
```

Csúszóátlag számítás parancssorból

Fejlesztés

- Újabb egyszerűsítés: ha újrahasznosítjuk a tömb egyes elemeit, akkor egy 3 elemű tömb is megteszi.

Csúszóátlag számítás parancssorból

csuszoatlag.c (v1.7) [1–29]

```c
/* Csúszóátlag számítása.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

double atlag3(double v1, double v2, double v3) {
    return (v1 + v2 + v3) / 3;
}

int main(int argc, char *argv[]) {
    double ertektomb[3];
    if (argc < 4) {
        return 1;
    }
    ertektomb[0] = atof(argv[1]);
    ertektomb[1] = atof(argv[2]);
    for (int i = 0; i < argc - 3; ++i) {
        ertektomb[2] = atof(argv[i + 3]);
        printf("%lf;", atlag3(ertektomb[0], ertektomb[1], ertektomb[2]));
        ertektomb[0] = ertektomb[1];
        ertektomb[1] = ertektomb[2];
    }
    putchar('\n');
    return 0;
}
```

Csúszóátlag számítás parancssorból

Fejlesztés

- Korrigáljuk az `i` ciklusváltozó tartományát.

Csúszóátlag számítás parancssorból

csuszoatlag.c (v2.0) [1–29]

```c
/* Csúszóátlag számítása.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

double atlag3(double v1, double v2, double v3) {
    return (v1 + v2 + v3) / 3;
}

int main(int argc, char *argv[]) {
    double ertektomb[3];
    if (argc < 4) {
        return 1;
    }
    ertektomb[0] = atof(argv[1]);
    ertektomb[1] = atof(argv[2]);
    for (int i = 3; i < argc; ++i) {
        ertektomb[2] = atof(argv[i]);
        printf("%lf;", atlag3(ertektomb[0], ertektomb[1], ertektomb[2]));
        ertektomb[0] = ertektomb[1];
        ertektomb[1] = ertektomb[2];
    }
    putchar('\n');
    return 0;
}
```

Tartalom

Basename standard inputról

basename.c (v1.0) [1–25]

```c
/* Adott egy (linuxos) útvonalleírás, nyerjük ki belőle
 * a fájl nevét (az útvonal nélkül).
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Szeptember 9. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

#define N 32

void basename(char path[], char base[]) {
    int i = 0, lastsep = -1;
    for (; path[i] != 0; ++i) {
        if (path[i] == '/') {
            lastsep = i;
        }
    }
    ++lastsep;
    i = 0;
    while (path[lastsep] != 0) {
        base[i++] = path[lastsep++];
    }
    base[i] = 0;
}
```

Basename standard inputról

basename.c (v1.0) [27–33]

```
27 int main() {
28     char path[N], base[N];
29     fgets(path, N, stdin);
30     basename(path, base);
31     fputs(base, stdout);
32     return 0;
33 }
```

Basename standard inputról

A probléma

- Egy linuxos útvonal leírásból határozzuk meg a fájl nevét, az elérési útvonal nélkül.
- Ez gyakorlatilag fájlleíró karaktersorozat utolsó része, amely nem tartalmazza a / karaktert.
- Az útvonalat bekérjük a felhasználótól.

Basename standard inputról

basename.c (v0.1) [1–12]

```c
/* Adott egy (linuxos) útvonalleírás, nyerjük ki belőle
 * a fájl nevét (az útvonal nélkül).
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Szeptember 9. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

int main() {
    return 0;
}
```

Sztringek C-ben [1/2]

- Karaktersorozatot egyszerűen egy karakteres tömbbel készítünk.

```
char str[h];
```

- A `char str[h]` változó számára h bájt foglalódik és maximum $h-1$ karakter hosszú szöveg tárolható benne. A szöveg maximális méretére (a fizikai korlátokon kívül) nincs korlátozás.
- Az `str` szöveg i. karakterére az `str[i-1]` változóhivatkozással hivatkozhatunk.
- A sztring értékeket idézőjelek között lehet megadni.

```
"Helló␣Világ!"
"Hány␣forintot␣váltsunk?"
"%lf"
"%lf␣HUF␣=␣%lf␣EUR\n"
```

Sztringek C-ben [2/2]

- A karaktertömb számára lefoglalt hely (a sztring *mérete*) valójában csak egy felső korlátot jelent a sztring *hosszára* nézve. A sztring aktuális értéke ettől a korláttól lefelé bármikor eltérhet.
- A szöveg végét a szöveghez tartozó utolsó karakter után elhelyezett `'\0'` karakter jelzi. Így egy h *méretű* karaktertömbben maximum $h-1$ értékes karaktert, ezáltal maximum $h-1$ karakter *hosszú* szöveget tárolhatunk.
- Legyen `char str[6];` a sztring deklarációja, ekkor az `"egy"`, `"alma"` és `"meggy"` szavak, valamint az üres sztring (`""`) az alábbi módon tárolódnak el.

0.	1.	2.	3.	4.	5.
e	g	y	\0		
a	l	m	a	\0	
m	e	g	g	y	\0
\0					

Sztringek megadása

- Sztring literálok használhatók inicializálásra, de értékadásra az = művelettel közvetlenül nem.
- Értékadásra az

```
#include <string.h>
```

sor megadása után az `strcpy()` függvény használható.

```
char str1[20] = "Helló␣Világ!";
char str2[]   = "Helló␣Világ!"; /* automatikus méret */
char str3[32];

/* str3= "Helló Világ!"; /* Ez hibás!! */
strcpy(str3, "Helló␣Világ!");
```

Basename standard inputról

basename.c (v0.2) [1–15]

```c
/* Adott egy (linuxos) útvonalleírás, nyerjük ki belőle
 * a fájl nevét (az útvonal nélkül).
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Szeptember 9. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

#define N 32

int main() {
    char path[N];
    return 0;
}
```

Standard be- és kimenet

`stdin`, `stdout`, `stderr`

- Minden programban létezik 3 eleve nyitott kommunikációs csatorna (fájl):

`stdin`	bemeneti csatorna
`stdout`	kimeneti csatorna
`stderr`	hiba csatorna

- A `scanf()` és `printf()` az `stdin` és `stdout` csatornákat (fájlokat) használják.
- Vannak általános függvények, amelyek nem kötöttek ezekhez a standard fájlokhoz, de használhatóak velük.
  - `fgets(str, size, file)`
    - Beolvas a `size` méretű `str` sztringbe egy sort a `file` fájlból.
  - `fputs(str, file)`
    - Kiírja a `file` fájlba az `str` sztringet.

Basename standard inputról

basename.c (v0.3) [1–17]

```c
/* Adott egy (linuxos) útvonalleírás, nyerjük ki belőle
 * a fájl nevét (az útvonal nélkül).
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Szeptember 9. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

#define N 32

int main() {
    char path[N];
    fgets(path, N, stdin);
    fputs(path, stdout);
    return 0;
}
```

Tömb mint paraméter

- Egy függvény paramétere lehet tömb típusú is.
- Mivel C-ben nincs indexhatár-ellenőrzés, és a paraméterként átadott tömbnek csak a címét kapja meg a függvény, ezért a paraméterként kapott egydimenziós tömb pontos méretét a függvény deklarációjában nem kell megadni.
- A függvény módosíthatja a tömb elemeit (ha azt máshogyan nem tiltjuk)!

Basename standard inputról

basename.c (v0.4) [1–26]

```c
/* Adott egy (linuxos) útvonalleírás, nyerjük ki belőle
 * a fájl nevét (az útvonal nélkül).
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Szeptember 9. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

#define N 32

void basename(char path[], char base[]) {
    int i = 0, lastsep = 0;
    while (path[lastsep] != 0) {
        base[i++] = path[lastsep++];
    }
    base[i] = 0;
}

int main() {
    char path[N], base[N];
    fgets(path, N, stdin);
    basename(path, base);
    fputs(base, stdout);
    return 0;
}
```

Basename standard inputról

Fejlesztés

- A függvény keresse meg az utolsó / karaktert.

Basename standard inputról

basename.c (v0.5) [1–24]

```c
/* Adott egy (linuxos) útvonalleírás, nyerjük ki belőle
 * a fájl nevét (az útvonal nélkül).
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Szeptember 9. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

#define N 32

void basename(char path[], char base[]) {
    int i = 0, lastsep = 0;
    for (; path[i] != 0; ++i) {
        if (path[i] == '/') {
            lastsep = i;
        }
    }
    i = 0;
    while (path[lastsep] != 0) {
        base[i++] = path[lastsep++];
    }
    base[i] = 0;
}
```

Basename standard inputról

Fejlesztés

- Korrigáljuk a hibát.

Basename standard inputról

basename.c (v1.0) [1–25]

```c
/* Adott egy (linuxos) útvonalleírás, nyerjük ki belőle
 * a fájl nevét (az útvonal nélkül).
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Szeptember 9. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

#define N 32

void basename(char path[], char base[]) {
    int i = 0, lastsep = -1;
    for (; path[i] != 0; ++i) {
        if (path[i] == '/') {
            lastsep = i;
        }
    }
    ++lastsep;
    i = 0;
    while (path[lastsep] != 0) {
        base[i++] = path[lastsep++];
    }
    base[i] = 0;
}
```

Basename standard inputról

basename.c (v1.0) [27–33]

```
27 int main() {
28     char path[N], base[N];
29     fgets(path, N, stdin);
30     basename(path, base);
31     fputs(base, stdout);
32     return 0;
33 }
```

Tartalom

Basename parancssorból

basename.c (v2.0) [1–26]

```c
/* Adott egy (linuxos) útvonalleírás, nyerjük ki belőle
 * a fájl nevét (az útvonal nélkül).
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Szeptember 9. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

const char * basename(const char *path) {
    const char *base = path;
    for (; *path != 0; ++path) {
        if (*path == '/') {
            base = path + 1;
        }
    }
    return base;
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
        return 1;
    }
    puts(basename(argv[1]));
    return 0;
}
```

Basename parancssorból

A probléma

- A probléma ugyanaz mint az előbb.
- Az inputot most a parancssorból vesszük.

Basename parancssorból

basename.c (v1.0) [1–25]

```c
/* Adott egy (linuxos) útvonalleírás, nyerjük ki belőle
 * a fájl nevét (az útvonal nélkül).
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Szeptember 9. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

#define N 32

void basename(char path[], char base[]) {
    int i = 0, lastsep = -1;
    for (; path[i] != 0; ++i) {
        if (path[i] == '/') {
            lastsep = i;
        }
    }
    ++lastsep;
    i = 0;
    while (path[lastsep] != 0) {
        base[i++] = path[lastsep++];
    }
    base[i] = 0;
}
```

Basename parancssorból

basename.c (v1.0) [27–33]

```
27 int main() {
28     char path[N], base[N];
29     fgets(path, N, stdin);
30     basename(path, base);
31     fputs(base, stdout);
32     return 0;
33 }
```

Basename parancssorból

Fejlesztés

- Vegyük az inputot a parancssorból . . .

Basename parancssorból

basename.c (v1.1) [27–32]

```
27 int main(int argc, char *argv[]) {
28     char base[N];
29     basename(argv[1], base);
30     fputs(base, stdout);
31     return 0;
32 }
```

Basename parancssorból

Fejlesztés

- . . . feltéve, hogy van elegendő (legalább egy) parancssori paraméter.
- Az `fgets()`-szel való beolvasással ellentétben a `main` argumentumaként kapott sztring végén nem lesz sortörés.

Basename parancssorból

basename.c (v1.2) [27–35]

```
27 int main(int argc, char *argv[]) {
28     if (argc < 2) {
29         return 1;
30     }
31     char base[N];
32     basename(argv[1], base);
33     puts(base);
34     return 0;
35 }
```

Basename parancssorból

Fejlesztés

- Figyeljünk rá, hogy már nincs kontrol alatt az input hossza!

Basename parancssorból

basename.c (v1.3) [1–25]

```c
/* Adott egy (linuxos) útvonalleírás, nyerjük ki belőle
 * a fájl nevét (az útvonal nélkül).
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Szeptember 9. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

#define N 32

void basename(char path[], char base[]) {
    int i = 0, lastsep = -1;
    for (; path[i] != 0; ++i) {
        if (path[i] == '/') {
            lastsep = i;
        }
    }
    ++lastsep;
    i = 0;
    while (path[lastsep] != 0 && i < N - 1) {
        base[i++] = path[lastsep++];
    }
    base[i] = 0;
}
```

Pointer típus

- Már a K&R is óvatosságra int:
  „*Azt szokták mondani, hogy a mutató, csakúgy, mint a `goto` utasítás, csak arra jó, hogy összezavarja és érthetetlenné tegye a programot. Ez biztos így is van, ha ész nélkül használjuk, hiszen könnyűszerrel gyárthatunk olyan mutatókat, amelyek valamilyen nem várt helyre mutatnak. Kellő önfegyelemmel azonban a mutatókat úgy is alkalmazhatjuk, hogy ezáltal programunk világos és egyszerű legyen.*”

Pointer típusképzés C-ben

- Egy adott típusú pointer értéke egy olyan memóriaterület címe, ahol a pointer típusa által adott típusú elem(ek) van(nak).
- Pointer típusú változót az alábbi módon deklarálhatunk:

```
char * pc;
unsigned short int * pi;
```

Pointerek és tömbök

- A C nyelvben szoros kapcsolat van a mutatók és a tömbök között.
  - Valamennyi művelet, amely tömbindexeléssel végrehajtható, mutatók használatával éppúgy elvégezhető.
  - Általában az utóbbi változat gyorsabb, de különösen a kezdők számára első ránézésre nehezebben érthető.
- A pointer által mutatott memóriaterületet elképzelhetjük úgy, mintha ott egy megfelelő tömb lenne.
- A tömb azonosítható egyetlen memóriacímmel, ahol a tömb adatai kezdődnek.
  - (Valójában pontosan ez történik, a tömb neve egy pointer érték.)
- Vagyis a pointer tulajdonképpen a tömb 0-s indexű elemére mutat, tehát a pointer dereferencia a nulladik tömbelem hivatkozással ekvivalens.

Pointerek és tömbök

- Amikor tömb vagy pointer adódik át valamelyik függvénynek, a függvény tetszése szerint hiheti azt, hogy tömböt vagy mutatót kapott (hiszen mindkettő egy memóriacím átadásával jár), és ennek megfelelően kezelheti azt.
- A függvény deklarációjában akár

```
f(int arg[]) { ... }
```

akár

```
f(int * arg) { ... }
```

is lehet.

Konstansok

A `const` módosító

- A `const` módosító hatására az adott változó (beleértve a függvényparamétert is, hiszen az is egyfajta változóként viselkedik a függvényben) módosíthatatlanságát a fordító ellenőrzi.
- Összetett deklaráció esetén a `const` a megelőző típus-egységre vonatkozik (illetve a legelsőre, ha nem előzi meg egy sem).
  - `const char * p`; `char const * p`:
    - konstansra mutató változtatható pointer
    - a mutatott karakter nem változhat, de a pointer igen
  - `char * const p`:
    - változóra mutató konstans pointer
    - a mutatott karakter változhat, de a pointer nem

Basename parancssorból

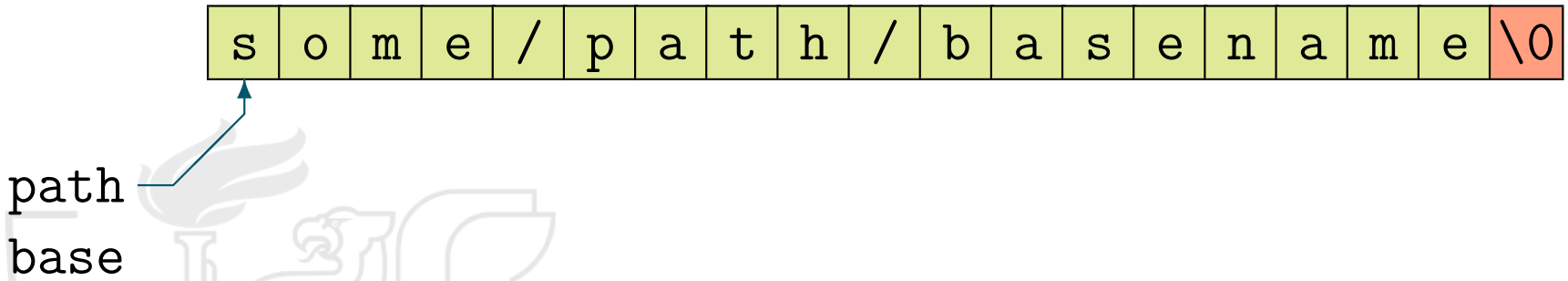
basename.c (v1.4) [1–25]

```c
/* Adott egy (linuxos) útvonalleírás, nyerjük ki belőle
 * a fájl nevét (az útvonal nélkül).
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Szeptember 9. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

#define N 32

void basename(const char * const path, char * const base) {
    int i = 0, lastsep = -1;
    for (; path[i] != 0; ++i) {
        if (path[i] == '/') {
            lastsep = i;
        }
    }
    ++lastsep;
    i = 0;
    while (path[lastsep] != 0 && i < N - 1) {
        base[i++] = path[lastsep++];
    }
    base[i] = 0;
}
```

Basename parancssorból

Fejlesztés

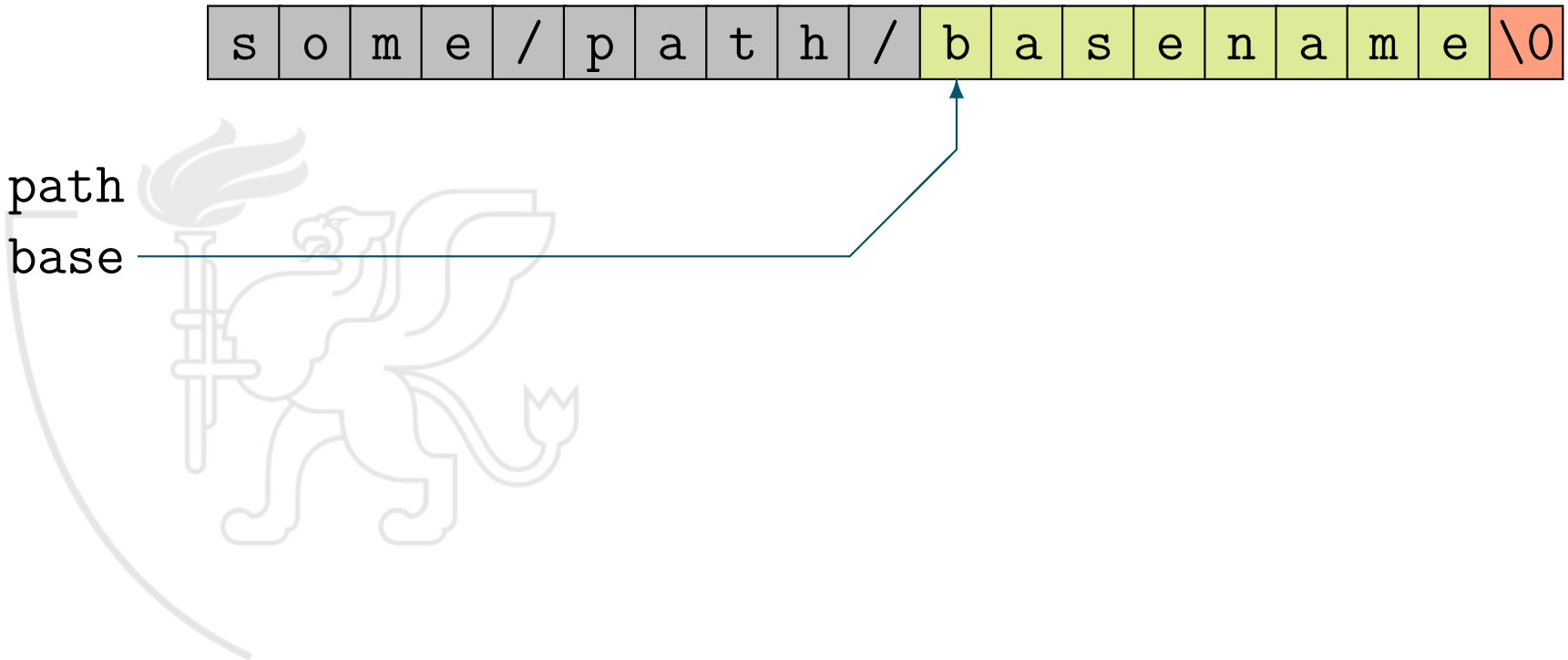
- Gondoljuk végig: a basename az eredeti útvonalleírás végén lévő sztring lesz.
- Teljesen felesleges lemásolni, csak meg kell mutatni, hol kezdődik az eredeti sztringben.



Basename parancssorból

Fejlesztés

- Gondoljuk végig: a basename az eredeti útvonalleírás végén lévő sztring lesz.
- Teljesen felesleges lemásolni, csak meg kell mutatni, hol kezdődik az eredeti sztringben.



Basename parancssorból

basename.c (v1.5) [1–26]

```c
/* Adott egy (linuxos) útvonalleírás, nyerjük ki belőle
 * a fájl nevét (az útvonal nélkül).
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Szeptember 9. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

const char * basename(const char * const path) {
    int lastsep = -1;
    for (int i = 0; path[i] != 0; ++i) {
        if (path[i] == '/') {
            lastsep = i;
        }
    }
    return &(path[lastsep + 1]);
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
        return 1;
    }
    puts(basename(argv[1]));
    return 0;
}
```

Pointerek és tömbök

- Pointeraritmetika: pointerhez hozzáadhatunk illetve belőle kivonhatunk egész számokat.
- A tömbindexelés és a pointeraritmetika között nagyon szoros kapcsolat van.
- Általában `ptr-i` a `ptr` előtti `i`. elem címe és `ptr+i` a `ptr` utáni `i`. elem címe, vagyis

```
*(ptr+i) ==   ptr[i];
  ptr+i  == &(ptr[i]);
```

- Ha `ptr` egy mutató, akkor
  - `ptr++` / `ptr--` oly módon inkrementálja / dekrementálja `ptr`-t, hogy az a megcímzett tetszőleges típusú objektum következő / előző elemére mutasson,
  - `ptr+=i` / `ptr-=i` pedig úgy inkrementálja / dekrementálja `ptr`-t, hogy az a pillanatnyilag megcímzett elem utáni / előtti `i`-edik elemre mutasson.

Basename parancssorból

basename.c (v2.0) [1–26]

```c
/* Adott egy (linuxos) útvonalleírás, nyerjük ki belőle
 * a fájl nevét (az útvonal nélkül).
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Szeptember 9. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

const char * basename(const char *path) {
    const char *base = path;
    for (; *path != 0; ++path) {
        if (*path == '/') {
            base = path + 1;
        }
    }
    return base;
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
        return 1;
    }
    puts(basename(argv[1]));
    return 0;
}
```

Tartalom

Tér legtávolabbi pontjai

ter.c (v1.0) [1–29]

```c
/* A háromdimenziós tér pontjai közül keressük meg a két legtávolabbit!
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Július 26. Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

typedef struct {              /* A tér egy háromdimenziós pontjának típusa */
    double x, y, z;
} pont_t;

pont_t read_pont() {
    pont_t p = {0.0, 0.0, 0.0};
    scanf("%lf %lf %lf", &p.x, &p.y, &p.z);
    return p;
}

void write_pont(pont_t p) {
    printf("(%lf, %lf, %lf)", p.x, p.y, p.z);
}

double tav(pont_t p, pont_t q) {
    return sqrt((p.x - q.x) * (p.x - q.x) +
                (p.y - q.y) * (p.y - q.y) +
                (p.z - q.z) * (p.z - q.z));
}
```

Tér legtávolabbi pontjai

ter.c (v1.0) [31–58]

```c
void legtavolabb(int n, pont_t p[], pont_t *egyik, pont_t *masik) {
    double max = -1.0;
    for (int i = 0; i < n - 1; ++i) {
        for (int j = i + 1; j < n; ++j) {
            if (tav(p[i], p[j]) > max) {
                max = tav(p[i], p[j]);
                *egyik = p[i];
                *masik = p[j];
            }
        }
    }
}

int main() {
    int darab, i;
    pont_t *tomb, p1, p2;
    scanf("%d", &darab);
    tomb = malloc(darab * sizeof(*tomb));
    for (i = 0; i < darab; ++i) {
        tomb[i] = read_pont();
    }
    legtavolabb(darab, tomb, &p1, &p2);
    printf("A␣legtávolabbi␣pontok:\n");
    printf("␣␣␣␣␣␣"); write_pont(p1); printf("␣és\n");
    printf("␣␣␣␣␣␣"); write_pont(p2); putchar('\n');
    free(tomb);
    return 0;
}
```

Tér legtávolabbi pontjai

A probléma

- Adott a térben valahány pont, melyik a két legtávolabbi?
- A pontok számát a pontok koordinátái előtt megkapjuk.

Tér legtávolabbi pontjai

ter.c (v0.1) [1–13]

```c
/* A háromdimenziós tér pontjai közül keressük meg a két legtávolabbit!
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Július 26. Gergely Tamás, gertom.inf.u-szeged.hu
 */

typedef struct {                /* A tér egy háromdimenziós pontjának típusa */
    double x, y, z;
} pont_t;

int main() {
    return 0;
}
```

Tér legtávolabbi pontjai

Fejlesztés

- Készítsünk függvényt a pontok beolvasására.
- Készítsünk függvényt a pontok kiírására.

Tér legtávolabbi pontjai

ter.c (v0.4) [1–28]

```c
/* A háromdimenziós tér pontjai közül keressük meg a két legtávolabbit!
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Július 26. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

typedef struct {                /* A tér egy háromdimenziós pontjának típusa */
    double x, y, z;
} pont_t;

pont_t read_pont() {
    pont_t p = {0.0, 0.0, 0.0};
    scanf("%lf %lf %lf", &p.x, &p.y, &p.z);
    return p;
}

void write_pont(pont_t p) {
    printf("(%lf, %lf, %lf)", p.x, p.y, p.z);
}

int main() {
    /* TESZT */ pont_t p;
    /* TESZT */ p = read_pont();
    /* TESZT */ write_pont(p); putchar('\n');
    return 0;
}
```

Tér legtávolabbi pontjai

Fejlesztés

- Szükség lesz két pont távolságára.

Tér legtávolabbi pontjai

ter.c (v0.6) [1–28]

```c
/* A háromdimenziós tér pontjai közül keressük meg a két legtávolabbit!
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Július 26. Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

typedef struct {              /* A tér egy háromdimenziós pontjának típusa */
    double x, y, z;
} pont_t;

pont_t read_pont() {
    pont_t p = {0.0, 0.0, 0.0};
    scanf("%lf %lf %lf", &p.x, &p.y, &p.z);
    return p;
}

void write_pont(pont_t p) {
    printf("(%lf, %lf, %lf)", p.x, p.y, p.z);
}

double tav(pont_t p, pont_t q) {
    return sqrt((p.x - q.x) * (p.x - q.x) +
                (p.y - q.y) * (p.y - q.y) +
                (p.z - q.z) * (p.z - q.z));
}
```

Tér legtávolabbi pontjai

ter.c (v0.6) [30–35]

```
30 int main() {
31     /* TESZT */ pont_t p1 = {3.0, 4.0, 0.0}, p2 = {0.0, 0.0, 12.0};
32     /* TESZT */ write_pont(p1); printf("␣és␣");
33     /* TESZT */ write_pont(p2); printf("␣távolsága␣%lf\n", tav(p1, p2));
34     return 0;
35 }
```

Tér legtávolabbi pontjai

Fejlesztés

- Egyelőre dolgozzunk fix számú ponttal.

Tér legtávolabbi pontjai

ter.c (v0.7) [1–24]

```c
/* A háromdimenziós tér pontjai közül keressük meg a két legtávolabbit!
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Július 26. Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

#define N 5

typedef struct {                /* A tér egy háromdimenziós pontjának típusa */
    double x, y, z;
} pont_t;

pont_t read_pont() {
    pont_t p = {0.0, 0.0, 0.0};
    scanf("%lf %lf %lf", &p.x, &p.y, &p.z);
    return p;
}

void write_pont(pont_t p) {
    printf("(%lf, %lf, %lf)", p.x, p.y, p.z);
}
```

Tér legtávolabbi pontjai

ter.c (v0.7) [26–42]

```
26 double tav(pont_t p, pont_t q) {
27     return sqrt((p.x - q.x) * (p.x - q.x) +
28                 (p.y - q.y) * (p.y - q.y) +
29                 (p.z - q.z) * (p.z - q.z));
30 }
31
32 int main() {
33     int i;
34     pont_t tomb[N];
35     for (i = 0; i < N; ++i) {
36         tomb[i] = read_pont();
37     }
38     for (i = 0; i < N; ++i) {
39         write_pont(tomb[i]);
40     }
41     return 0;
42 }
```

Cím szerinti paraméterátadás

- C-ben a függvények értékeket vesznek át, ez azt jelenti, hogy ha a függvényen belül meg is változik a paraméter értéke, az az argumentumra nincs hatással.
- Ha azt szeretnénk, hogy egy függvény megváltoztassa az argumentumként kapott változó értékét, trükkhöz kell folyamodnunk:
  - A paraméter `t par` deklarációja helyett `t *par` pointer-deklarációt alkalmazunk.
  - A függvénytörzsben `par` helyett mindenhol `(*par)` változóhivatkozást használunk.
  - A függvény meghívásakor az `arg` argumentum helyett az `&arg` argumentumot használjuk.
- Vagyis a változó értéke helyett a változóra mutató pointert várjuk paraméterként, és a változó címét adjuk át argumentumként.

Tér legtávolabbi pontjai

ter.c (v0.8) [26–47]

```
26 double tav(pont_t p, pont_t q) {
27     return sqrt((p.x - q.x) * (p.x - q.x) +
28                 (p.y - q.y) * (p.y - q.y) +
29                 (p.z - q.z) * (p.z - q.z));
30 }
31
32 void legtavolabb(pont_t p[], pont_t *egyik, pont_t *masik) {
33     /* TESZT */ *egyik = *masik = p[0];
34 }
35
36 int main() {
37     int i;
38     pont_t tomb[N], p1, p2;
39     for (i = 0; i < N; ++i) {
40         tomb[i] = read_pont();
41     }
42     legtavolabb(tomb, &p1, &p2);
43     printf("A␣legtávolabbi␣pontok:\n");
44     printf("␣␣␣␣␣␣"); write_pont(p1); printf("␣és\n");
45     printf("␣␣␣␣␣␣"); write_pont(p2); putchar('\n');
46     return 0;
47 }
```

Tér legtávolabbi pontjai

Fejlesztés

- A legtávolabbi pontok meghatározásához minden pontot minden ponttal össze kell hasonlítani . . .

Tér legtávolabbi pontjai

ter.c (v0.9) [32–54]

```
32 void legtavolabb(pont_t p[], pont_t *egyik, pont_t *masik) {
33     for (int i = 0; i < N; ++i) {
34         for (int j = 0; j < N; ++j) {
35             if (/* TESZT */ tav(p[i], p[j]) > -1.0) {
36                 *egyik = p[i];
37                 *masik = p[j];
38             }
39         }
40     }
41 }
42
43 int main() {
44     int i;
45     pont_t tomb[N], p1, p2;
46     for (i = 0; i < N; ++i) {
47         tomb[i] = read_pont();
48     }
49     legtavolabb(tomb, &p1, &p2);
50     printf("A␣legtávolabbi␣pontok:\n");
51     printf("␣␣␣␣␣␣"); write_pont(p1); printf("␣és\n");
52     printf("␣␣␣␣␣␣"); write_pont(p2); putchar('\n');
53     return 0;
54 }
```

Tér legtávolabbi pontjai

Fejlesztés

- . . . miközben kiválasztjuk az összehasonlítások közül a legnagyobb távolságot (és a hozzá tartozó két pontot) . . .

Tér legtávolabbi pontjai

ter.c (v0.10) [32–56]

```
32 void legtavolabb(pont_t p[], pont_t *egyik, pont_t *masik) {
33     double max = -1.0;
34     for (int i = 0; i < N; ++i) {
35         for (int j = 0; j < N; ++j) {
36             if (tav(p[i], p[j]) > max) {
37                 max = tav(p[i], p[j]);
38                 *egyik = p[i];
39                 *masik = p[j];
40             }
41         }
42     }
43 }
44
45 int main() {
46     int i;
47     pont_t tomb[N], p1, p2;
48     for (i = 0; i < N; ++i) {
49         tomb[i] = read_pont();
50     }
51     legtavolabb(tomb, &p1, &p2);
52     printf("A␣legtávolabbi␣pontok:\n");
53     printf("␣␣␣␣␣␣"); write_pont(p1); printf("␣és\n");
54     printf("␣␣␣␣␣␣"); write_pont(p2); putchar('\n');
55     return 0;
56 }
```

Tér legtávolabbi pontjai

Fejlesztés

- . . . de saját magával egyik pontot sem hasonlítjuk, és a pontpárokat is csak egyszer (az egyik irányban).

Tér legtávolabbi pontjai

ter.c (v0.11) [32–56]

```
32 void legtavolabb(pont_t p[], pont_t *egyik, pont_t *masik) {
33     double max = -1.0;
34     for (int i = 0; i < N - 1; ++i) {
35         for (int j = i + 1; j < N; ++j) {
36             if (tav(p[i], p[j]) > max) {
37                 max = tav(p[i], p[j]);
38                 *egyik = p[i];
39                 *masik = p[j];
40             }
41         }
42     }
43 }
44
45 int main() {
46     int i;
47     pont_t tomb[N], p1, p2;
48     for (i = 0; i < N; ++i) {
49         tomb[i] = read_pont();
50     }
51     legtavolabb(tomb, &p1, &p2);
52     printf("A␣legtávolabbi␣pontok:\n");
53     printf("␣␣␣␣␣␣"); write_pont(p1); printf("␣és\n");
54     printf("␣␣␣␣␣␣"); write_pont(p2); putchar('\n');
55     return 0;
56 }
```

Tér legtávolabbi pontjai

Fejlesztés

- A pontkeresést készítsük fel tetszőleges számú pontra.

Tér legtávolabbi pontjai

ter.c (v0.12) [32–57]

```
32 void legtavolabb(int n, pont_t p[], pont_t *egyik, pont_t *masik) {
33     double max = -1.0;
34     for (int i = 0; i < n - 1; ++i) {
35         for (int j = i + 1; j < n; ++j) {
36             if (tav(p[i], p[j]) > max) {
37                 max = tav(p[i], p[j]);
38                 *egyik = p[i];
39                 *masik = p[j];
40             }
41         }
42     }
43 }
44
45 int main() {
46     int darab, i;
47     pont_t tomb[N], p1, p2;
48     scanf("%d", &darab);
49     for (i = 0; i < darab; ++i) {
50         tomb[i] = read_pont();
51     }
52     legtavolabb(darab, tomb, &p1, &p2);
53     printf("A␣legtávolabbi␣pontok:\n");
54     printf("␣␣␣␣␣␣"); write_pont(p1); printf("␣és\n");
55     printf("␣␣␣␣␣␣"); write_pont(p2); putchar('\n');
56     return 0;
57 }
```

Pointer típus műveletei

Pointer műveletek

`*p`	p által mutatott érték.
`&v`	v változó címe.
`NULL`	„nem mutat sehová” érték.*
`malloc(s)`	s méretű memóriaterületet lefoglalása.*
`free(p)`	p által mutatott memóriaterületet felszabadítása.*

```
long *p;
 p = NULL;
 p = malloc(sizeof(*p));
*p = 5;
free(p);
 p = &v;
```

*: használatához szükséges az

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

Tér legtávolabbi pontjai

ter.c (v1.0) [1–29]

```c
/* A háromdimenziós tér pontjai közül keressük meg a két legtávolabbit!
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Július 26. Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

typedef struct {              /* A tér egy háromdimenziós pontjának típusa */
    double x, y, z;
} pont_t;

pont_t read_pont() {
    pont_t p = {0.0, 0.0, 0.0};
    scanf("%lf %lf %lf", &p.x, &p.y, &p.z);
    return p;
}

void write_pont(pont_t p) {
    printf("(%lf, %lf, %lf)", p.x, p.y, p.z);
}

double tav(pont_t p, pont_t q) {
    return sqrt((p.x - q.x) * (p.x - q.x) +
                (p.y - q.y) * (p.y - q.y) +
                (p.z - q.z) * (p.z - q.z));
}
```

Tér legtávolabbi pontjai

ter.c (v1.0) [31–58]

```c
void legtavolabb(int n, pont_t p[], pont_t *egyik, pont_t *masik) {
    double max = -1.0;
    for (int i = 0; i < n - 1; ++i) {
        for (int j = i + 1; j < n; ++j) {
            if (tav(p[i], p[j]) > max) {
                max = tav(p[i], p[j]);
                *egyik = p[i];
                *masik = p[j];
            }
        }
    }
}

int main() {
    int darab, i;
    pont_t *tomb, p1, p2;
    scanf("%d", &darab);
    tomb = malloc(darab * sizeof(*tomb));
    for (i = 0; i < darab; ++i) {
        tomb[i] = read_pont();
    }
    legtavolabb(darab, tomb, &p1, &p2);
    printf("A␣legtávolabbi␣pontok:\n");
    printf("␣␣␣␣␣␣"); write_pont(p1); printf("␣és\n");
    printf("␣␣␣␣␣␣"); write_pont(p2); putchar('\n');
    free(tomb);
    return 0;
}
```

Tartalom

A nappalis gyakorlat értékelése [1/10]

eredmeny.c [1–23]

```c
/* Programozás alapjai gyakorlat értékelés.
 *
 * A program a parancssorban sorban megadott zh és plusz pontok alapján
 * kiszámolja a várható érdemjegyet.
 *
 * 2023. Május 22.  Gergely Tamás, gertom@inf.u-szeged.hu
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>

#ifndef SILENT
#define GR "A gyakorlat eredménye: "
#else
#define GR
#endif
#define G1 "elégtelen(1)"
#define G2 "elégséges(2)"
#define G3 "közepes(3)"
#define G4 "jó(4)"
#define G5 "jeles(5)"
```

A nappalis gyakorlat értékelése [2/10]

eredmeny.c [25–43]

```c
#define MAX_ZH_PER_WEEK 2

#define OPT_HELP_S "-h"
#define OPT_HELP_L "--help"
#define OPT_NO_MIN "--no-minimums"

typedef struct {
    int   min;
    char *jegy;
} eredmeny_t;

const eredmeny_t G[] = {
    {80, G5},
    {70, G4},
    {57, G3},
    {45, G2},
    { 0, G1},
    { 0, NULL}
};
```

A nappalis gyakorlat értékelése [3/10]

eredmeny.c [45–61]

```
45 const int ZHPWEEK[] = {  1,  1,  1,  1,  1,  1,  1,  1,  2,  0 };
46 const int MINIMUM[] = {  1,  4,  7, 10, 15, 20, 25, 30, 45 };
47 const int ZHMAXPT[] = {  5,  5,  5,  5, 10, 10, 10, 10, 10, 10 };
48 const int MAXPLUS   = 5;
49 const int MAXPRGP   = 5;
50 const int MAXPSUM   = 10;
51 const int NINCSMIN  = 1;
52 const int NINCSVED  = 1;
53
54 bool minimum_ellenorzes = true;
55 int  vedes_nem_sikerult = 0;
56
57 int heti_pont(int w, int zhn, int zhp[], int hfp[]) {
58     int sum = 0;
59     int db = ZHPWEEK[w];
60     bool zhnemved = false;
61     bool hfnemved = false;
```

A nappalis gyakorlat értékelése [4/10]

eredmeny.c [63–90]

```
63      for (int i = 0, j = 0; i < db; ++i, ++zhn) {
64          if (zhp[i] < 0) {
65              zhnemved = true;
66              zhp[i] = 0;
67          }
68          if (hfp[i] < 0) {
69              hfnemved = true;
70              hfp[i] = 0;
71          }
72          sum += ((zhp[i] >= ZHMAXPT[zhn]) ? ZHMAXPT[zhn] : zhp[i]) + (!!hfp[j++]);
73      }
74
75      if (zhnemved) {
76  #ifndef SILENT
77          printf("Figyelmeztetés! A(z) %d. zh védése nincs elfogadva.\n", w + 1);
78  #endif
79          ++vedes_nem_sikerult;
80      }
81
82      if (hfnemved) {
83  #ifndef SILENT
84          printf("Figyelmeztetés! A(z) %d. házi feladat védése nincs elfogadva.\n", w + 1);
85  #endif
86          ++vedes_nem_sikerult;
87      }
88
89      return sum;
90  }
```

A nappalis gyakorlat értékelése [5/10]

eredmeny.c [92–110]

```
92 bool min_bukas(int w, int zhp) {
93     static int nincs_min = 0;
94     if (minimum_ellenorzes && zhp < MINIMUM[w]) {
95 #ifndef SILENT
96         printf("Figyelmeztetés! A(z) %d. zh után nincs meg a minimális pontszám.\n"
97                "    elvárt: %d, meglévő: %d.\n", w + 1, MINIMUM[w], zhp);
98 #endif
99         if (++nincs_min > NINCSMIN) {
100 #ifndef SILENT
101             printf("A minimum pontszám kritériumait legalább %d alkalommal"
102                    " nem teljesítette, ami több, mint az elfogadható %d alkalom.\n",
103                    nincs_min, NINCSMIN);
104 #endif
105             printf(GR G1 "\n");
106             return true;
107         }
108     }
109     return false;
110 }
```

A nappalis gyakorlat értékelése [6/10]

eredmeny.c [112–135]

```
112 void usage(const char* progname) {
113     int db;
114     int maxpont = 0;
115     printf("Használat: %s [" OPT_HELP_S "|" OPT_HELP_L "|" OPT_NO_MIN "]", progname);
116     for (int w = 0; (db = ZHPWEEK[w]); ++w) {
117         if (db == 1) {
118             printf(" hf%d zh%d", w + 1, w + 1);
119         } else {
120             for (int i = 0; i < db; ++i) {
121                 printf(" hf%d/%d zh%d/%d", w + 1, i + 1, w + 1, i + 1);
122             }
123         }
124     }
125     printf(" plusz pprog\n"
126            "\n"
127            "    hfN: az N. heti házi feladat pontjai\n"
128            "    hfN/M: az N. heti zh M. házi feladat pontjai\n"
129            "    zhN: az N. heti zh pontszáma\n"
130            "    zhN/M: az N. heti zh M. részpontszáma\n"
131            "    plusz: plusz pontszám\n"
132            "    pprog: pluszprogram pontszám\n"
133            "\n"
134            "Összpontszámok minimuma (a plusz és pprog nem számítanak bele):\n"
135            "\n");
```

A nappalis gyakorlat értékelése [7/10]

eredmeny.c [136–161]

```
136     for (int w = 0, tmp = 0; (db = ZHPWEEK[w]); ++w) {
137         maxpont += db;
138         for (int i = 0; i < db; ++i) {
139             maxpont += ZHMAXPT[tmp++];
140         }
141         printf("␣␣␣␣%d.␣hét␣után␣%d␣pont␣(az␣addigi␣összesen␣%d␣pontból)\n",
142                w + 1, MINIMUM[w], maxpont);
143     }
144     maxpont += MAXPSUM;
145 
146     printf("\n"
147            "Ha␣a␣minimum␣pontszámot␣több,␣mint␣%d␣alkalommal"
148            "␣nem␣sikerül␣teljesíteni,\n"
149            "vagy␣a␣leadott␣feladatok␣közül␣több,␣mint␣%d␣darabot"
150            "␣nem␣sikerül␣megvédeni,\n"
151            "akkor␣a␣félév␣végeredménye␣minden␣mástól␣függetlenül␣" G1 ".\n"
152            "\n"
153            "Értékelés,␣az␣összes␣részpont,␣plusz␣(maximum␣%d)"
154            "␣és␣pprog␣(maximum␣%d,␣összesen␣max␣%d)␣összege␣alapján:\n"
155            "\n",
156            NINCSMIN, NINCSVED, MAXPLUS, MAXPRGP, MAXPSUM);
157     for (const eredmeny_t *p = G; p->jegy; ++p) {
158         printf("␣␣␣␣%3d␣--␣%3d␣pont:␣%s\n", p->min, maxpont, p->jegy);
159         maxpont = p->min - 1;
160     }
161 }
```

A nappalis gyakorlat értékelése [8/10]

eredmeny.c [163–186]

```
163 int main(int argc, char *argv[]) {
164     int sum = 0;
165     int plus = 0;
166     int prgp = 0;
167     int idx = 1;
168     int db;
169     int w, zhn;
170 
171     if (argc < 2) {
172 #ifndef SILENT
173         usage(argv[0]);
174 #endif
175         return 1;
176     }
177 
178     if (!strcmp(OPT_HELP_S, argv[1]) || !strcmp(OPT_HELP_L, argv[1])) {
179         usage(argv[0]);
180         return 0;
181     }
182 
183     if (!strcmp(OPT_NO_MIN, argv[1])) {
184         minimum_ellenorzes = false;
185         idx = 2;
186     }
```

A nappalis gyakorlat értékelése [9/10]

eredmeny.c [188–215]

```
188     for (w = 0, zhn = 0; (db = ZHPWEEK[w]) && (idx < argc); ++w, zhn += db) {
189         int hfp[MAX_ZH_PER_WEEK];
190         int zhp[MAX_ZH_PER_WEEK];
191         for (int i = 0; i < db; ++i) {
192             hfp[i] = (idx < argc) ? atoi(argv[idx]) : 0;
193             ++idx;
194             zhp[i] = (idx < argc) ? atoi(argv[idx]) : 0;
195             ++idx;
196         }
197         if (idx <= argc) {
198             if (min_bukas(w, sum += heti_pont(w, zhn, zhp, hfp))) {
199                 return 0;
200             }
201         } else {
202             break;
203         }
204     }
205
206     if (vedes_nem_sikerult > NINCSVED) {
207#ifndef SILENT
208         printf("A␣leadott␣feladata␣%d␣alkalommal␣nem␣lett␣elfogadva,␣ami␣több,"
209                "␣mint␣az␣elfogadható␣%d␣alkalom.\n"
210                "A(z)␣%d␣pontját␣így␣nem␣tudjuk␣figyelembe␣venni.\n",
211                vedes_nem_sikerult, NINCSVED, sum);
212#endif
213         printf(GR G1 "\n");
214         return 0;
215     }
```

A nappalis gyakorlat értékelése [10/10]

eredmeny.c [217–245]

```
217     if (db > 0) {  // Ha még nincs pontszám minden zh-hoz (break volt)
218#ifndef SILENT
219         printf("%d␣zh␣eredménye␣alapján␣jelenleg␣%d␣pontja␣van.\n", w, sum);
220#endif
221         return 0;
222     }
223
224     if (idx < argc) { // ha van plusz pont is
225         plus = atoi(argv[idx++]);
226     }
227     plus = (plus > MAXPLUS) ? MAXPLUS : plus;
228     if (idx < argc) { // ha van program plusz pont is
229         prgp = atoi(argv[idx++]);
230     }
231     plus += (prgp > MAXPRGP) ? MAXPRGP : prgp;
232
233     sum += (plus > MAXPSUM) ? MAXPSUM : plus;
234
235#ifndef SILENT
236     printf("A␣végső␣pontszáma%s␣%d.\n", (idx < argc) ? "␣legalább" : "", sum);
237#endif
238     for (const eredmeny_t *p = G; p->jegy; ++p) { // A pontszámhoz tartozó jegy keresése
239         if (sum >= p->min) {
240             printf(GR "%s\n", p->jegy);
241             return 0;
242         }
243     }
244     return -1;
245 }
```