Programozás 1

A C nyelv alapjai

A C nyelv

- 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
- Viszont gazdagon kínál standard függvénykönyvtárakat.

A C nyelv tulajdonságai

- Kis méret
- Függvényhívások széleskörű használata
- A típus hanyagolása
- Struktúrált nyelv
- Alacsony szintű programozás olvasható elérése
- Pointer széleskörű használata:
  - memória, tömb, struktúra, függvény

A C nyelv tulajdonságai

- Miért vált a C széleskörűen használt programozási nyelvvé?
  - 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 rövid története

- UNIX fejlesztése 1969 körül az AT&T Bell Labratóriumában
  - Ken Thomson és Dennis Ritchie
  - Első változat Assembly nyelven DEC PDP-7 gépen.
- BCPL – felhasználóbarát operációs rendszer fejlesztő nyelv
  - Legfőbb jellemzője, hogy típustalan.
  - Unalmasan hosszú kód sok hibával.
- Egy új nyelv, a "B", a második kísérlet 1970 körül

A C nyelv rövid története

- 1971: teljesen új nyelv, a "C", a "B" leszármazottja
  - 1973-ban a UNIX operációs rendszert átírták "C"-re.
  - Az AT&T "hivatalos" változata a System V (System Five), amely a 4. Release-nél tart, rövidítve SVR4)
- 1988-89: ANSI C
  - American National Standards Institute
- 1999: ISO C 99
  - International Organization for Standardization

C fájlok

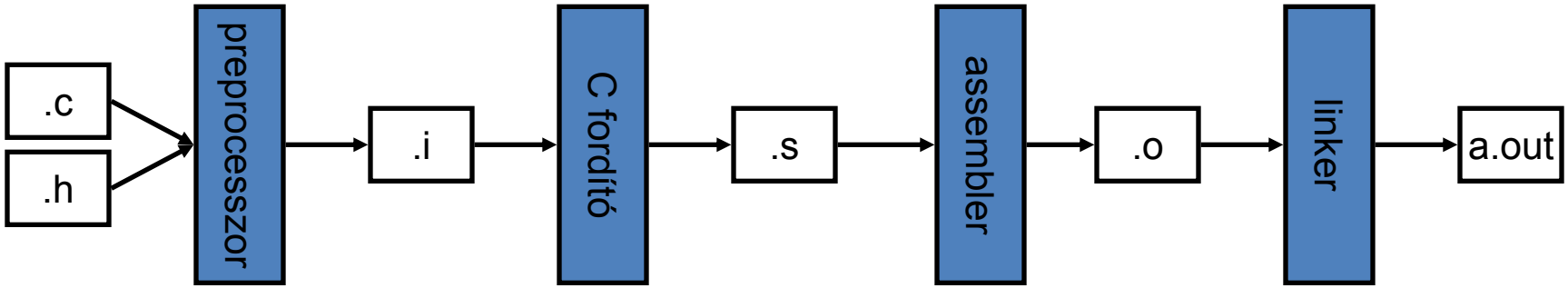
- A szabályos végződések:
  - **`file.c`** C source (forrás) file
  - **`file.h`** C header (fejléc) file
  - **`file.i`** preprocessed C (preprocesszált) file
  - **`file.s`** assembly language file
  - **`file.o`** object file
  - **`a.out`** link edited output

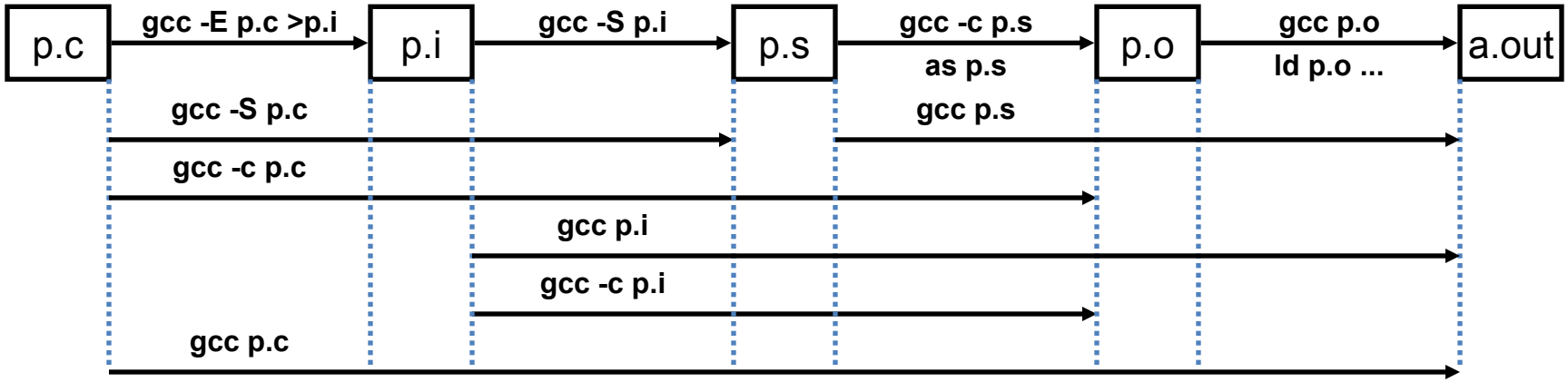
A C forrás fordításának folyamata

- A fordítási folyamat sok lépésből is állhat, de 4 olyan lépés van, ahol a folyamatot elkezdeni illetve befejezni lehet.
  - preprocessing – előfeldolgozás
  - compilation – fordítás (assembly nyelvre)
  - assembly – fordítás (gépi kódra)
  - linking – szerkesztés

A C forrás fordításának folyamata

- A fájl végződése utal a programozási nyelvre és arra, hogy mit kell vele csinálni:





Minimális C program

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


main()
{

}
```

Hello World C nyelven

```
/*
 * Helló Világ!
 * Készítette: Gergely Tamás, gertom@inf.u-szeged.hu
 *             2004. November 3.
 */
#include <stdio.h>

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

A C nyelv alapjai

- Egy C programozási nyelven írt forrásfájl programegységek deklarációjából és definíciójából áll.

- Deklarálhatunk
  - Típust
  - Változót
  - Függvényt

```
typedef unsigned long ui32;
```

```
int magassag;
```

```
double mertani_kozep(double, double);
```

Deklaráció

- Egy programkomponens deklarációja egy azonosító hozzárendelése az adott komponenshez. Ezzel az azonosítóval hivatkozhatunk a program további részében az adott komponensre.
- A program egy adott pontján csak azok a komponensek használhatók (hivatkozhatók), amelyeket e pontot megelőzően már deklaráltunk, ellenkező esetben fordítási hiba lép fel.

Definíció

- Egy programkomponens definíciója egy (a komponens deklarációjában meghatározott típusú) érték hozzárendelése a komponens azonosítójához.
- 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.

Adattípus

- Az adattípus olyan egysége a programnak, amely két összetevője által meghatározott:
  - Értékhalmaz
  - Az értékhalmaz elemein végezhető műveletek
- Minden adattípus vagy elemi, vagy más adattípusokból képzett összetett adattípus.

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 változó értéke kezdetben definiálatlan, és az marad, amíg valamilyen művelettel értéket nem adunk neki.

Változó

- A változók alapvetően úgynevezett értékadó művelet végrehajtásával kaphatnak értéket.
- Ennek jelölése:

  V = E

  ahol
  - V egy meghatározott adattípusú változó
  - E ezen adattípus egy értéke
- Feltételezhetjük, hogy az értékadó művelet minden elemi adattípus műveletei között szerepel.

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

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.
- Például:
  - `2` – egész, de
  - `2.0` – valós (lebegőpontos) lesz!

Konstans

- A C preprocesszor segítségével definiálhatunk konstansokat az alábbi *direktívával*:

  `#define azonosító érték`

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

- Preprocesszor direktívák a ***fordítás előtt*** feldolgozásra kerülnek.
  - A fenti direktíva hatására a forráskódban előforduló **`azonosító`**-t lecseréli a megadott **`érték`**-re
  - A fordító azután az így előállt (módosított) forráskódot kapja meg feldolgozásra!

Függvény

- A függvény a matematikai értelemben vett függvény általánosítása, gyakorlatilag egy (rész-)algoritmus megvalósítása.
- A függvény deklarációja adja meg, hogy milyen típusú értékekből milyen típusú értéket állít elő.
- A függvény definíciója megmondja, hogy a függvény végrehajtása során milyen algoritmust kell végrehajtani.
- A függvény meghívásakor azt adjuk meg, hogy milyen konkrét értékeken hajtsuk végre a függvény által leírt algoritmust.

Függvény

- A C nyelvben a függvény fogalma bővebb (~*eljárás*, *szubrutin*), mint a matematikában
  - Vannak olyan függvények, amiknek valamilyen jól definiált „mellékhatása” is van a visszatérési érték kiszámítása mellett, ilyenek például a szöveget megjelenítő (pl. **`printf`**), vagy adatbevitelt kezelő (pl. **`scanf`**) függvények.
  - Adott esetben az is előfordulhat, hogy egy függvénynek csak a „mellékhatása” fontos, ilyenkor lehet, hogy a függvény (matematikai értelemben) nem is számít ki semmilyen visszatérési értéket. Az ilyen függvényeket nevezhetjük eljárásnak.

Deklarációk, definíciók C-ben

- Változók deklarációja:

```
típusmegadás azonosító;
```

```
int i;
double f;
```

Szintaxis

- A kommunikáció ember és gép között véges jelhalmazból (ábécé) vett, meghatározott formai szabályokat kielégítő, véges hosszúságú jelsorozatokkal történik. Ezek a jelsorozatok alkotják a kommunikáció nyelvét.
- Szintaxis
  - Formai szabályok olyan rendszerét, amely meghatározza, hogy egy adott kommunikációs nyelvben melyek a szabályos jelsorozatok, a nyelv szintaxisának nevezzük.

Szintaxisdiagram

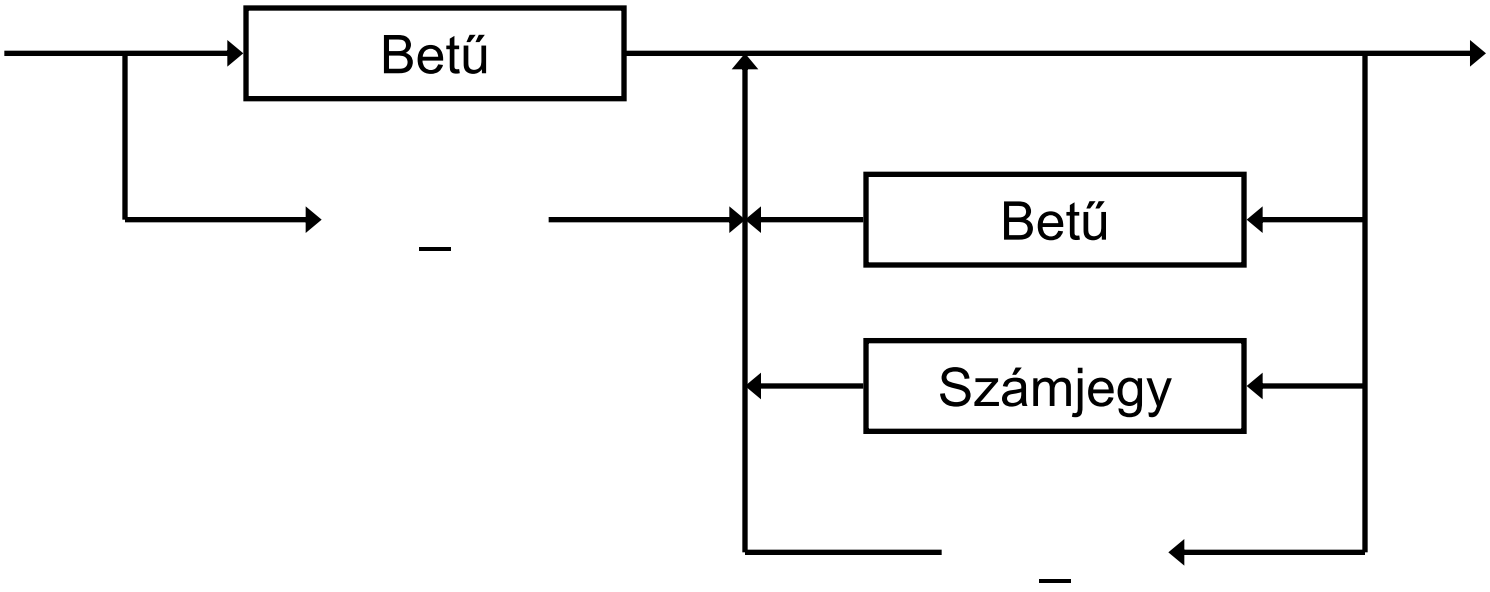
- Szintaxis megadására számos módszer ismeretes, mi szintaxis diagramokat használunk.
  - Ebben minden szintaktikus egység egyedi elnevezést kap, és a szintaktikus egységhez tartozó szabályos jelsorozatokat egy diagram (ábra) definiálja
  - Az ábrában a szintaktikus egységneveket tartalmazó dobozokat (téglalapokat) és konkrét jelsorozatokat irányított vonalak kötik össze
  - Minden diagramnak egy bemenete és egy kimenete van

Szintaxisdiagram

- Szintaxis diagramok egy (véges) rendszerében egy diagram azokat és csak azokat a jelsorozatokat határozza meg, amelyek úgy kaphatók, hogy a diagram bemenetéről indulva az irányított vonalak mentén haladva a kijáratig, valahányszor érintünk egy egységet, egymás után írjuk az úton érintett egység által meghatározott jelsorozatok egy elemét.
- A diagramban a konkrét jelsorozatok önmagukat definiálják.

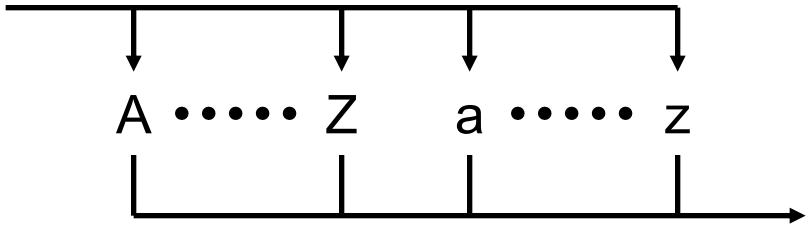
Szintaxis

- Azonosító

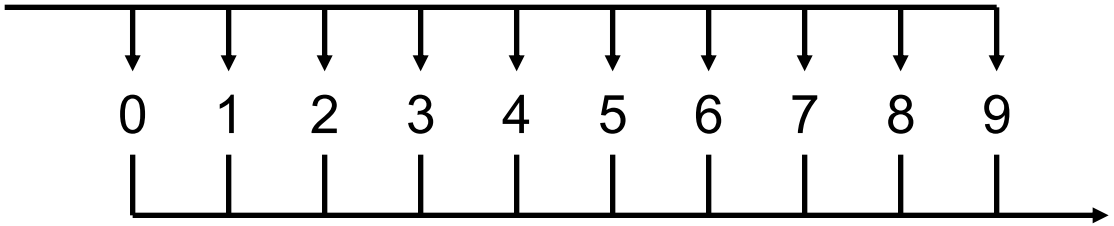


Szintaxis

- Betű



- Számjegy



A C nyelv elemi adattípusai

- **`int`**
  - Egész
- **`float`**
  - Valós

- A C-ben nincs logikai típus, de logikai *igaz* és *hamis* értékek keletkeznek, és logikai műveletek is vannak

Logikai műveletek

- Egy operandusú műveletek (logikai -> logikai)
  - **!** Tagadás művelet

```
!a
```

- Két operandusú műveletek (logikai x logikai -> logikai)
  - **&&** 'és' művelet
  - **||** 'vagy' művelet

```
a && b
```

```
a || b
```

Logikai műveletek

- Relációs műveletek
  (logikai x logikai -> logikai)
  - `==` Egyenlőség
  - `!=` Nemegyenlő
  - `<` Kisebb
  - `>` Nagyobb
  - `<=` Kisebb vagy egyenlő
  - `>=` Nagyobb vagy egyenlő

```
a == b
a != b
a < b
a > b
a <= b
a >= b
```

Az int adattípus

- A C nyelv eleve definiált elemi adattípusa.
- Egész értékek tárolására alkalmas.
- Értékkészlete
  - **`[INT_MIN, INT_MAX]`** zárt intervallumba eső egész számok halmaza
- Az **`INT_MIN`** és **`INT_MAX`** a **`limits.h`** header fájlban definiált konstans azonosítók,
  - használatukhoz a program elejére be kell szúrni az alábbi preprocesszor direktívát, amely betölti a **`limits.h`** header fájl tartalmát közvetlenül az **`#include`** direktíva helyére:

```
#include <limits.h>
```

Az int adattípus műveletei

- Egy operandusú műveletek (**`int`** -> **`int`**)
  - **`-`** Előjel váltás

```
-a
```

- Két operandusú műveletek (**`int`** x **`int`** -> **`int`**)
  - **`+`** Összeadás
  - **`-`** Kivonás
  - **`*`** Szorzás
  - **`/`** Egészosztás hányadosa
  - **`%`** Egészosztás maradéka

```
a + 8
```

```
7 - a
```

```
6 * 7
```

```
a / 3
```

```
42 % a
```

Az int adattípus műveletei

- Relációs műveletek
  (**`int`** x **`int`** -> logikai)
  - **`==`** Egyenlőség
  - **`!=`** Nem egyenlő
  - **`<`** Kisebb
  - **`>`** Nagyobb
  - **`<=`** Kisebb vagy egyenlő
  - **`>=`** Nagyobb vagy egyenlő

```
3 == a
```

```
a != 7
```

```
a < 42
```

```
a > 77
```

```
0 <= a
```

```
8 >= a
```

Az int adattípus műveletei

- Az **`int`** adattípus műveleteire teljesülnek az aritmetika ismert azonosságai, feltéve, hogy a művelet eredménye az adattípus értékhalmazába esik.

```
2 * (3 * 7 + 5 * 4) == 4 * 2 * 5 + 3 * 2 * 7
```

- Ha a művelet eredménye nem esne az adattípus értékhalmazába, túlcsordulásról beszélünk.

```
2147483647 + 1 == -2147483648
```

A float adattípus

- A C nyelv eleve definiált elemi adattípusa.
- A matematikai valós számok és műveleteik számítógépes modellezésére használható.
  - A matematikai valós számok megközelíthetők az adattípus értékeivel,
  - a matematikai műveletek pedig az adattípus műveleteivel.
  - A közelítés mértéke a konkrét gépi megvalósítástól függ.
- A **`float`** adattípust valós típusnak is hívjuk.

A float adattípus

- Értékkészlete
  - Az adattípus (diszkrét!) értékei egy adott intervallumba esnek úgy, hogy bármely valós szám ebből a folytonos intervallumból adott relatív pontossággal megközelíthető **`float`** adattípusbeli értékkel.
  - Ez azt jelenti, hogy
    - bármely a valós számhoz
    - van olyan x float típusú érték, hogy
    - $|(x-a)/a|$ < relatív pontosság

A float adattípus műveletei

- Egy operandusú műveletek (**`float`** -> **`float`**)
  - **`-`** Előjel váltás

```
-3.1415
```

- Két operandusú műveletek (**`float`** x **`float`** -> **`float`**)
  - **`+`** Összeadás
  - **`-`** Kivonás
  - **`*`** Szorzás
  - **`/`** Osztás

```
41.3 + 0.7
```

```
a - 2.7172
```

```
0.5 * a
```

```
a / 2.0
```

A float adattípus műveletei

- Relációs műveletek
  (**`float`** x **`float`** -> logikai)
  - **`==`** Egyenlőség
  - **`!=`** Nem egyenlő
  - **`<`** Kisebb
  - **`>`** Nagyobb
  - **`<=`** Kisebb vagy egyenlő
  - **`>=`** Nagyobb vagy egyenlő

```
3.0 == a
```

```
a != 7.2
```

```
a < 1e-10
```

```
a > 0.2E3
```

```
0.33 <= a
```

```
42.0 >= a
```

A float adattípus műveletei

- A matematikai függvények és konstansok, mint például a
  - **sin**
  - **cos**
  - **M_PI**
  - **log**
  - **exp**

```
double sin(double x);
```

```
double cos(double x);
```

```
#define M_PI 3.14159265358979323846
```

```
double log(double x);
```

```
double exp(double x);
```

nem részei a nyelvnek, de egy könyvtárban össze vannak gyűjtve. Használatukhoz szükséges a matematikai függvénykönyvtár include-olása

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

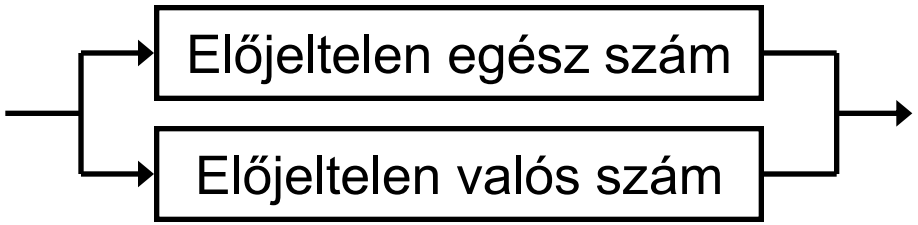
Numerikus adattípusok

- Az **`int`** és a **`float`** adattípusokat összefoglalóan numerikus adattípusoknak nevezzük.
- Numerikus adattípus értékeinek leírására számleírást, röviden számokat használunk.
  - Az **`int`** adattípus értékeit nyolcas, tízes vagy tizenhatos számrendszerbeli leírással adhatunk meg. (Tizenhatos számrendszerben a számjegyek értéke decimálisan: A=10, B=11, C=12, D=13, E=14, F=15.)
  - Valós számok leírására tizedes törtet használhatunk, kiegészítve tízes exponenssel.

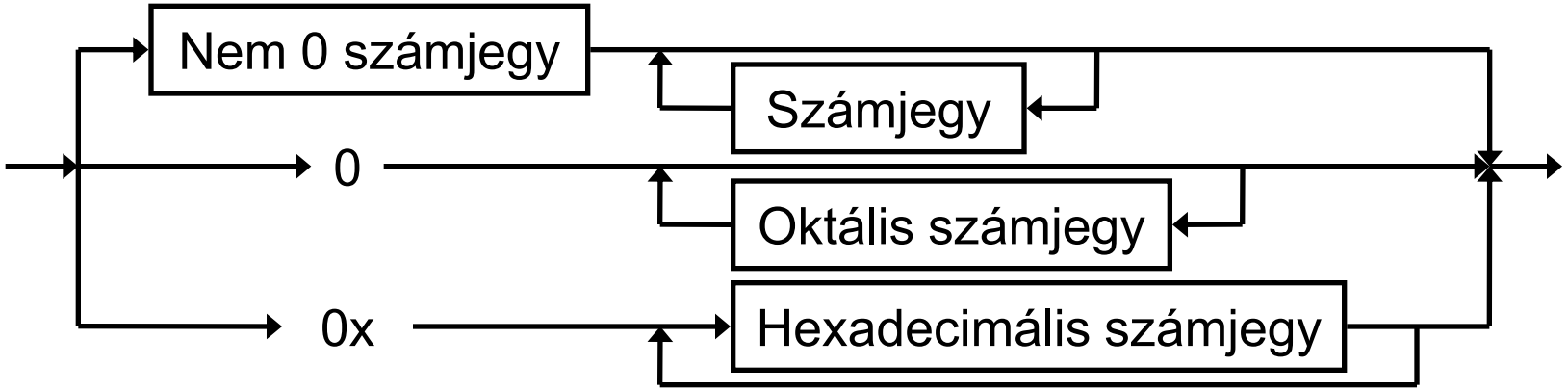
Szintaxis

- Szám

- Előjeltelen szám

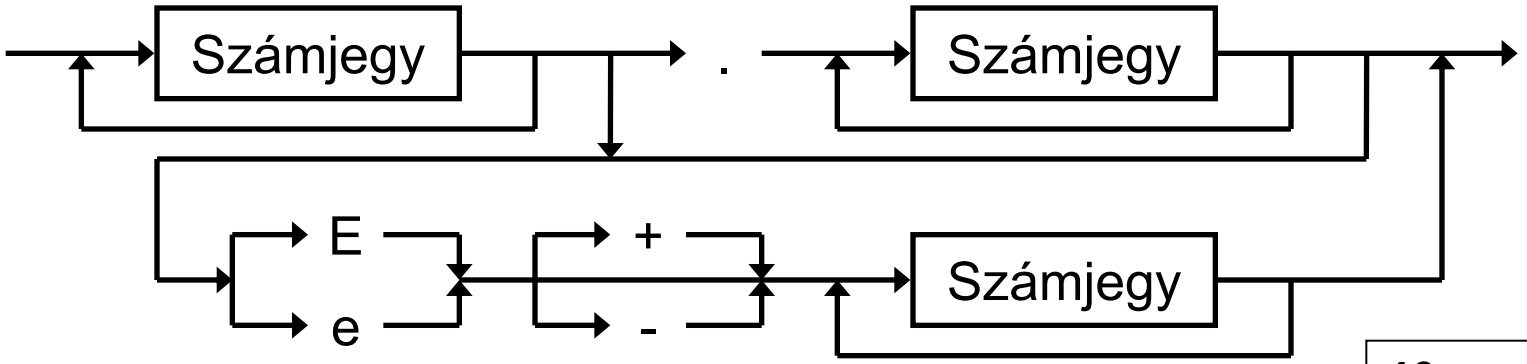


- Előjeltelen egész szám

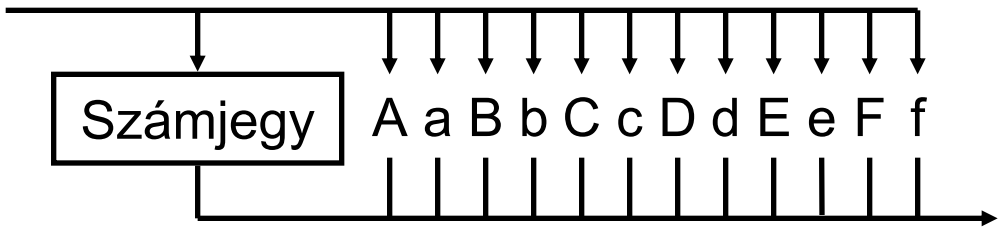


Szintaxis

- Előjeltelen valós szám



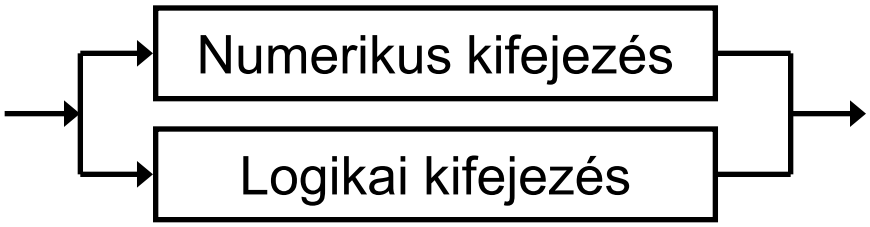
- Hexadecimális számjegy




```
42
052
0x2a
-3.1415
3e5
3E+6
271.72e-2
```

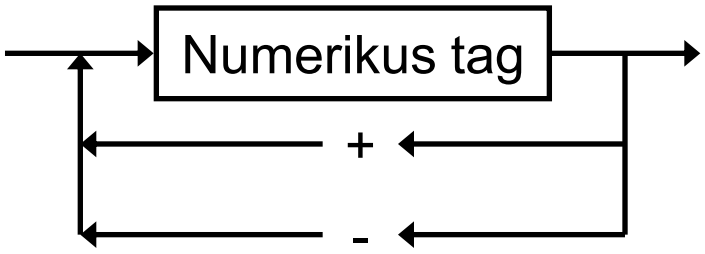
Kifejezés

- Kifejezésen olyan programkomponenst értünk, amely egy adattípus értékének olyan jelölése, amely műveleteket is tartalmazhat.
- A kifejezés által jelölt értéket a kifejezés kiértékelése határozza meg.
- A kifejezés szintaxisa (egyelőre)

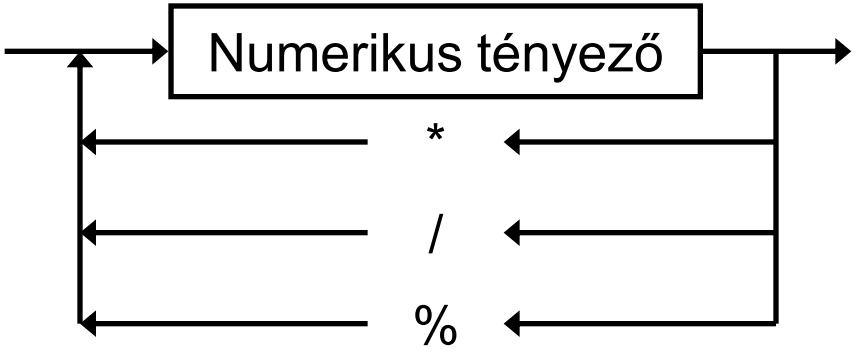


Numerikus kifejezés szintaxisa

- Numerikus kifejezés

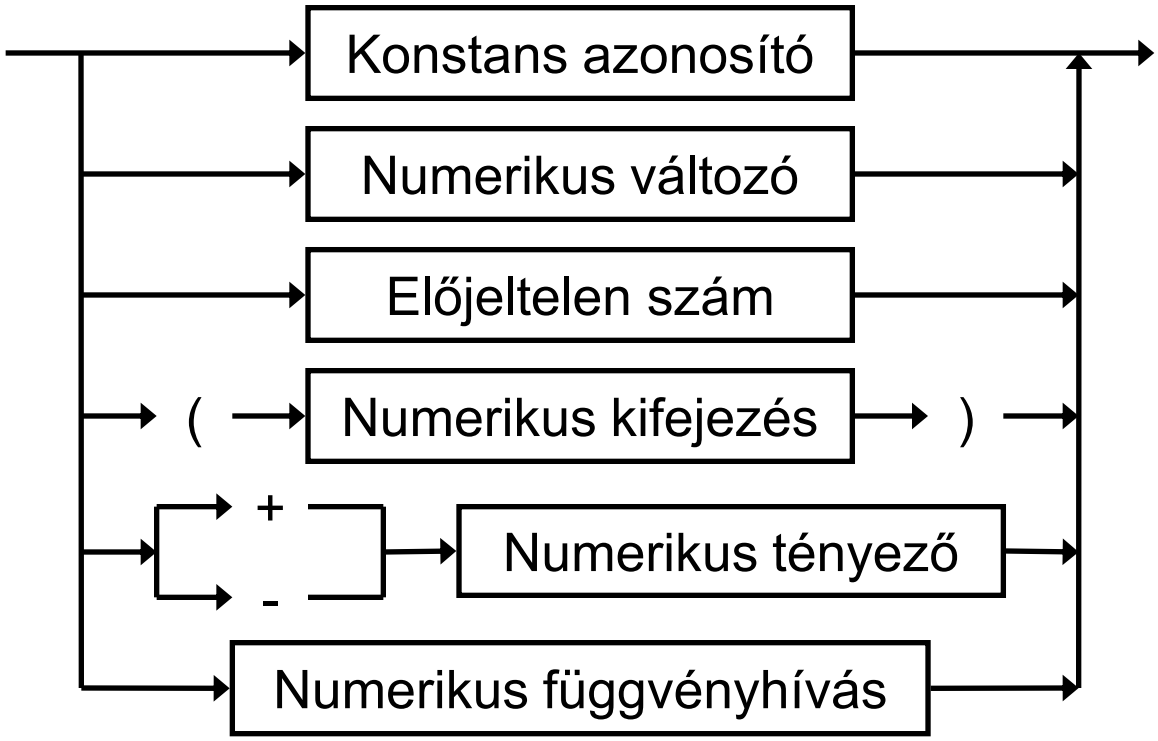


- Numerikus tag



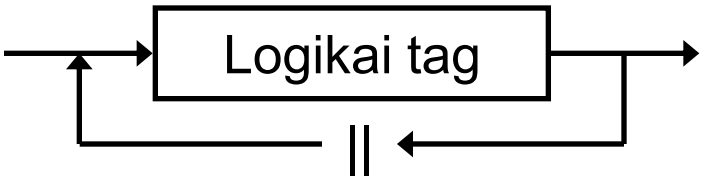
Numerikus kifejezés szintaxisa

- Numerikus tényező



Logikai kifejezés szintaxisa

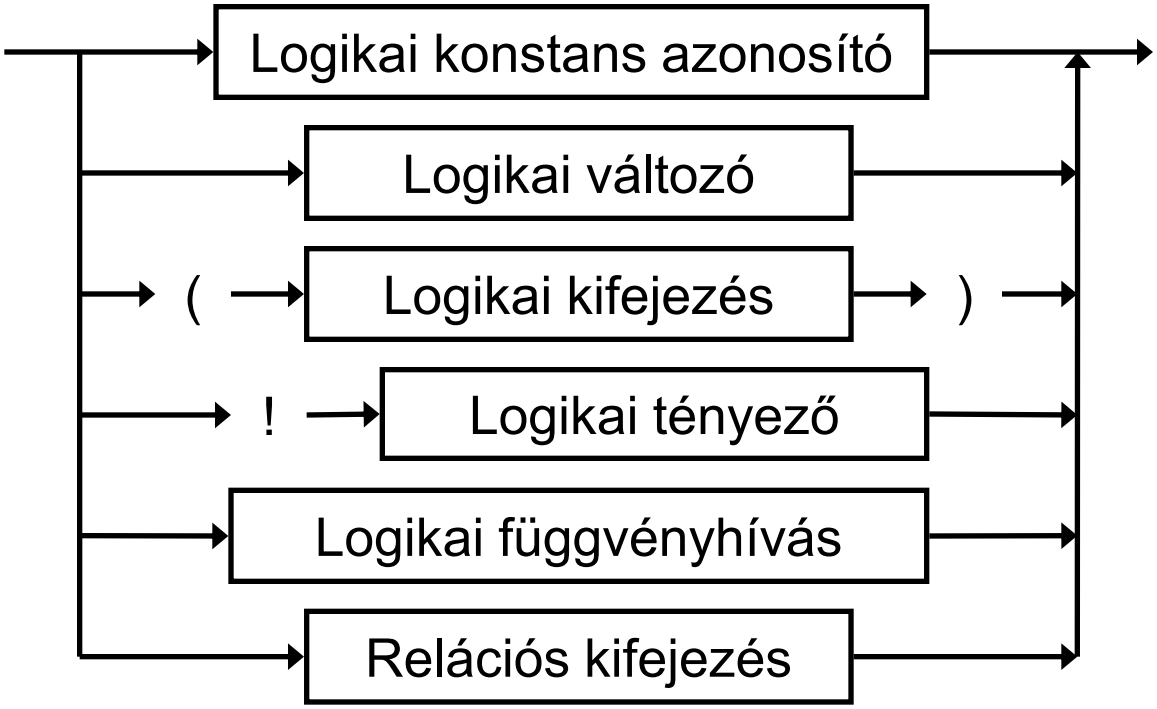
- Logikai kifejezés



- Logikai tag

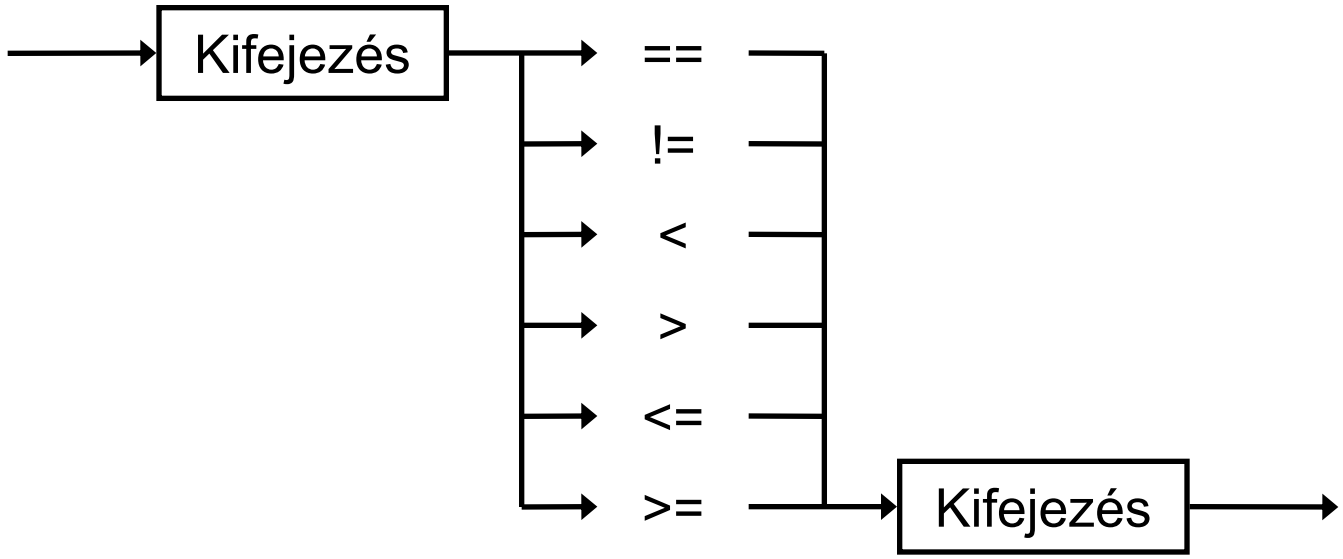
Logikai kifejezés szintaxisa

- Logikai tényező



Relációs kifejezés szintaxisa

- Relációs kifejezés




```
(sin(M_PI + alfa) < 0.2 * alfa + 1e-1) && (beta != M_PI)
```

Kifejezések kiértékelése

- A kifejezés kiértékelését két előírás együttesen határozza meg
  - A kifejezés szerkezetén alapuló prioritási előírás
  - A haladás irányára vonatkozó előírás

Prioritási előírás

- A műveletek között definiálva van egy erősségi sorrend, az úgynevezett prioritás.
- Ez azt jelenti, hogy egy A m_1 B m_2 C alakú kifejezésben, ahol m_2 magasabb prioritású művelet, mint m_1, az A m_1(B m_2 C) zárójelezésnek megfelelően értékelődik ki.

```
  a < b  &&   a +   - 5 * b    < 7   ||  c != b
((a < b) && ((a + ((- 5) * b)) < 7)) || (c != b)
```

Műveletek prioritása

- Prioritás csökkenő sorrendben:
  - a prefix művelet (prefix -, prefix +, !)
  - a multiplikatív műveletek (*, /, %)
  - az additív műveletek (+, -)
  - a kisebb-nagyobb relációs műveletek (<=, >=, <, >)
  - az egyenlő-nem egyenlő műveletek (==, !=)
  - a logikai 'és' művelet (&&)
  - a logikai 'vagy' művelet (||)

Műveletek asszociativitása

- Azonos prioritású műveletek esetén a kiértékelést az asszociativitás iránya szerint kell elvégezni.
- Ez azt jelenti, hogy egy A m_1 B m_2 C alakú kifejezés, ahol m_1 és m_2 azonos prioritású műveletek, a
  - balról-jobbra asszociativitás esetén (A m_1 B) m_2 C
  - jobbról-balra asszociativitás esetén A m_1 (B m_2 C)

  zárójelezésnek megfelelően értékelődik ki.
- Az eddig ismertetett műveletek balról-jobbra asszociatívak.

```
   a + 5  - b  - 7  + c
(((a + 5) - b) - 7) + c
```

Logikai kifejezések

- 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 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 lesz.

Numerikus kifejezések típusa

- Minden numerikus kifejezés **`int`** vagy **`float`** (vagy ezek változatai, ld. később) típusú.
- A kifejezés típusának meghatározása a kifejezés felépítése szerinti indukcióval a következő
  - Tényező típusa: ha a tényező
    - változó, akkor a deklarációjában megadott típus
    - konstans, akkor a számleírás szerinti típus (2 ⇔ 2.0)
    - (K), akkor a K kifejezés típusa
    - +T , akkor a T tényező típusa
    - -T , akkor a T tényező típusa
    - függvényhívás, akkor a függvényművelet eredménytípusa

Numerikus kifejezések típusa

- Tag típusa: legyen A m B alakú a tag, ahol A és B tényezők, m pedig multiplikatív műveleti jel. Ha m
  - `%`, akkor A és B típusa csak **`int`** lehet, és a tag típusa is **`int`**
  - **`/`**, és A és B típusa is **`int`**, akkor az eredmény is **`int`**, és a művelet az egészosztás hányadosa
  - **`/`**, és A vagy B **`float`**, akkor a tag típusa is **`float`**, és a művelet a valós osztás
  - **`*`**, akkor az eredmény tag típusa csak akkor lesz **`int`**, ha mindkét operandus típusa **`int`**, egyébként pedig **`float`**
- Kifejezés típusa: ha a tagok mindegyike **`int`** típusú, akkor a kifejezés típusa is **`int`**, egyébként **`float`**.

A / művelet

- A / lehet maradékos osztás de valós osztás is

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

Típus konverziók

- Az **`int`** adattípus nem része a **`float`** adattípusnak.
- **`float`** típusú változó vagy konstans akkor sem szerepelhet olyan műveletben, amely csak az **`int`** típus művelete (`%`), ha az értéke egész számértek.
- Megengedett azonban, hogy a **`float`** típus egy műveletének egyik argumentuma **`float`**, a másik argumentuma pedig **`int`** típusú legyen. A művelet eredményének típusa ekkor **`float`** lesz.

Típus konverziók

- Az ilyen művelet végrehajtása előtt az **`int`** típusú argumentum automatikusan átkonvertálódik **`float`** típusúvá. Ez a konverzió azonban tényleges műveletet jelent és így időt igényel.
- Ajánlatos a **`2*x`** művelet helyett **`2.0*x`** műveletet használni, ha `x` **`float`** típusú (még akkor is, ha a fordítóprogramok az ilyen típusú konverziót már fordítási időben el tudják végezni).

Értékadó művelet

- Az értékadás jele az **=**
- De ez művelet, és nem utasítás, vagyis a

  `változóazonosító = kifejezés`

  művelet eredménye a kifejezés aktuális értéke, amit a megfelelő programkomponensben is eltárolunk.
- Természetesen nincs akadálya a művelet többszöri alkalmazásának

```
i = j = k = 1;
```

- Az **=** művelet jobb-asszociatív, prioritása az eddig ismertetett műveletek után következik.

Utasítások

- A `;` zárja le műveletek sorozatát, tehát „utasítást csinál a kifejezésből”.
- A C-ben szokásos példa:
  - kifejezés `i = 1`
  - utasítás `i = 1;`

Összetett utasítások képzése

- Tudunk tehát kifejezésekből egyszerű utasításokat gyártani.
- Most mélyebb magyarázat nélkül átnézzük, hogy a C nyelv milyen lehetőségeket ad összetett utasítások képzésére.

Utasítások sorozata

- Ha az utasításokat adott sorrendben kell végrehajtani, akkor egyszerűen **{** és **}** jelek között az adott sorrendben egymás után leírjuk őket.
  - Számoljuk ki, melyik órában (o), percben (p) melyik másodperc (m) lesz a nap 54321. másodperce.

```
{
  e = 54321;
  o = e / 3600;
  e = e % 3600;
  p = e / 60;
  m = e % 60;
}
```

Függvények

- Egy C nyelven írt program tulajdonképpen nem más, mint függvények (megfelelően struktúrált és rendezett) összessége.
- Egy-egy függvény valamilyen bemenő adatok alapján kiszámol egy értéket, mint ahogyan azt a matematikában már megszokhattuk.
- Függvényeket lehet deklarálni, definiálni és meghívni.

Függvények

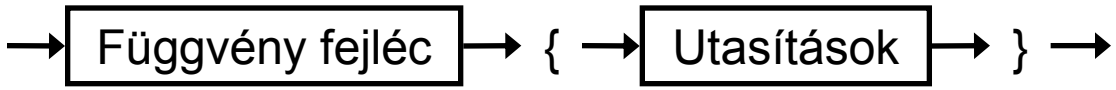
- Deklarációnál csak azt mondjuk meg, hogy mennyi és milyen típusú paraméterekből milyen típusú értéket fog kiszámolni a függvényünk.
- Definíciónál azt is meg kell adnunk, hogy hogyan számoljon.
- A függvényhívásnál pedig konkrét értékekre alkalmazzuk a függvényt, és a kiszámított értéket felhasználhatjuk további számolásainkhoz.

Függvények szintaxisa C-ben

- Függvény deklaráció

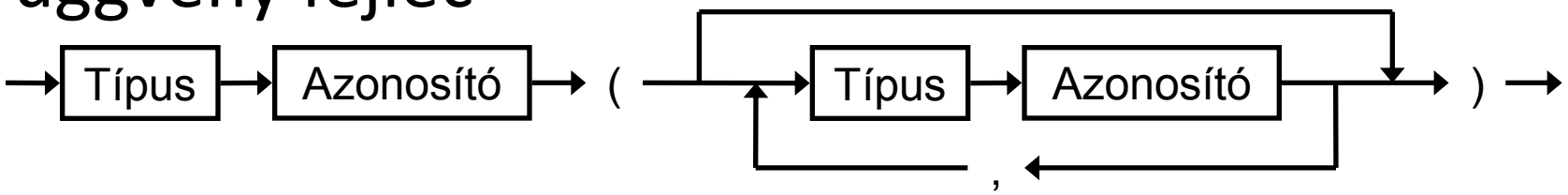

```
int f(int a, int b);
```

- Függvény definíció (egyben deklaráció is)




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

- Függvény fejléc



A `return` utasítás

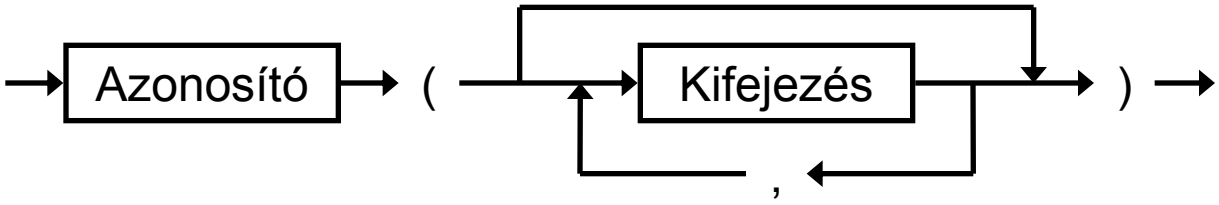
- A **`return`** utasítás


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

- 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.

Függvényhívás szintaxisa C-ben

- Függvényhívás



- Természetesen egy függvénynek a híváskor pontosan annyi és olyan típusú paramétert kell átadni, amennyi és amilyen paraméterekkel deklarálva lett.

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

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 speciális paramétereket.
- A **`main`** által kiszámított (egész szám) értéket szintén az operációs rendszer értelmezi (miután befejeződött a program).

Egy C program felépítése

- Egy egyszerű C program így néz ki:

```
/* A program adatai */

#include <stdio.h>

main() {
    Változódeklarációk
    Utasítások
}
```

Egy C program felépítése

- Egy kevésbé egyszerű pedig így:

```
/* A program adatai */

#include <stdio.h>

Függvénydefiníciók

main() {
    Változódeklarációk
    Utasítások
}
```

Függvények

- Írjunk egy programot, ami óra, perc, másodperc alapján egy függvény segítségével kiszámolja az éjfél óta eltelt másodperceket.

```
int masodpercek(int ora, int perc, int masodperc)
{
  return 3600 * ora + 60 * perc + masodperc;
}

int main()
{
  int mp1, mp2;
  mp1 = masodpercek(12, 5, 7);
  mp2 = masodpercek(6, 45, 0);
}
```

EGYSZERŰ I/O

Egyszerű ki- és bevitel

- Ahhoz, hogy egyszerű példát mutassunk be, szükségünk van beviteli és kiviteli utasításokra is.
- A nyelv nem tartalmaz ilyen utasításokat, de minden implementációban vannak standard függvénykönyvtárak ezek megvalósításaival.

Egyszerű ki- és bevitel

- A ki- és bevitel használatához szükségünk van az

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

  sorra a program elején
- Ezek után használhatjuk az alábbi két függvényt:
  - **`scanf(const char *, ...)`**
    - a bemenetről tudunk olvasni
  - **`printf(const char *, ...)`**
    - a kimenetre tudunk írni

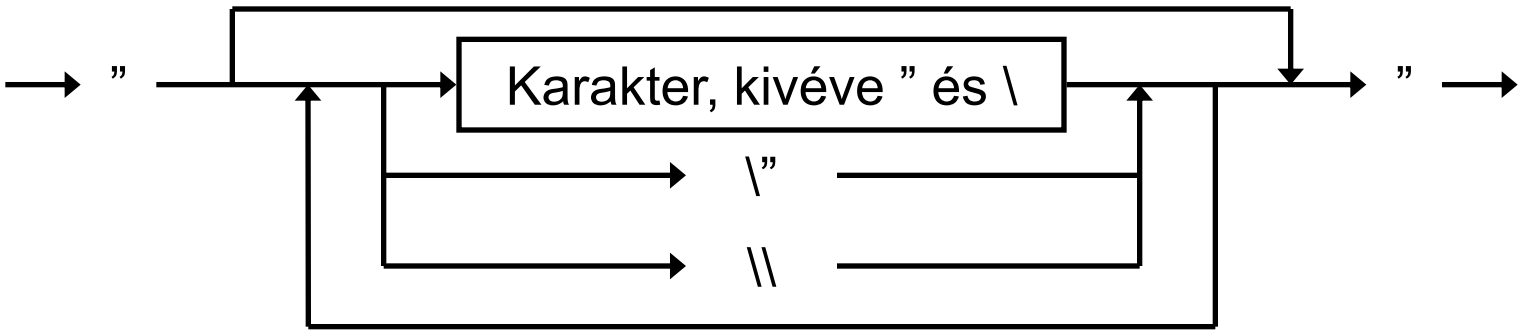
Egyszerű ki- és bevitel

- Anélkül, hogy a két függvényt részletesen elmagyaráznánk, egyelőre megmutatjuk, hogyan lehet **`int`** illetve **`float`** értékek beolvasására, valamint ugyanilyen típusú értékek és tetszőleges szöveg kiíratására használni őket.

Egyszerű ki- és bevitel

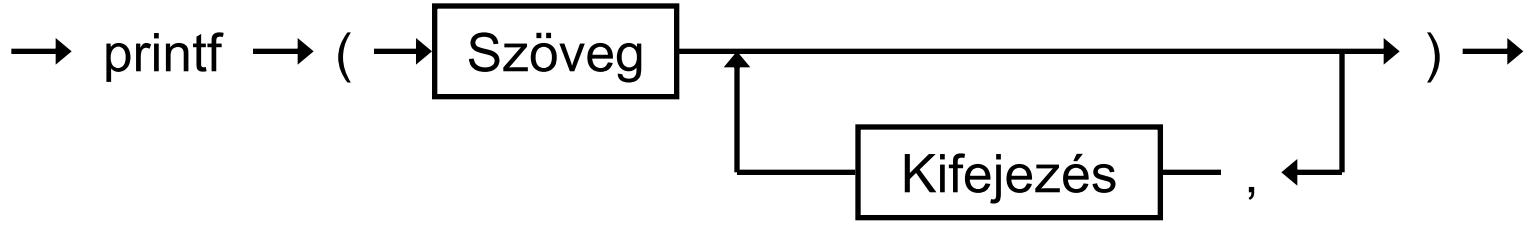
- Mindkét függvény első paramétere egy úgynevezett formátumsztring, ami tulajdonképpen egy speciális szövegkonstans.

- Szöveg (szövegkonstans):



Egyszerű ki- és bevitel

- A **`printf`** használata:



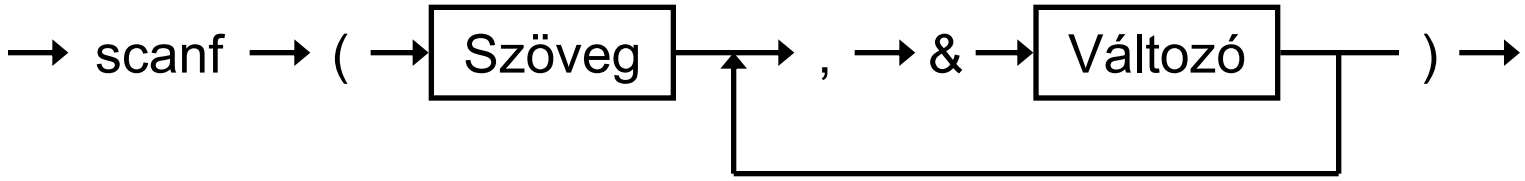
- A szövegben az egyes kifejezések helyét a **`%d`** (**`int`**) és **`%f`** (**`float`**) karakterkombinációk jelzik:

```
printf("Hello world\n");
printf("Pi értéke kb. %f\n", 3.1415926);
printf("%d + %d = %d\n", 2, 3, 2+3);
```

- A **`\n`** hatására a kiírás végén új sor kezdődik

Egyszerű ki- és bevitel

- A **`scanf`** használata:



- Az egyes beolvasandó számok típusát szintén **`%d`** (**`int`**) és **`%f`** (**`float`**) karakterkombinációk jelzik:

```
scanf("%d", &egesz);
scanf("%f %f", &valos1, &valos2);
```

Egyszerű ki- és bevitel

- Nagyon fontos, hogy a beolvasandó értékek illetve a kiírandó kifejezések számát és típusát sorban és pontosan adjuk meg. Egy **`float`** 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 **`%f`** –et.
- Az alábbi példák tehát hibásak:

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