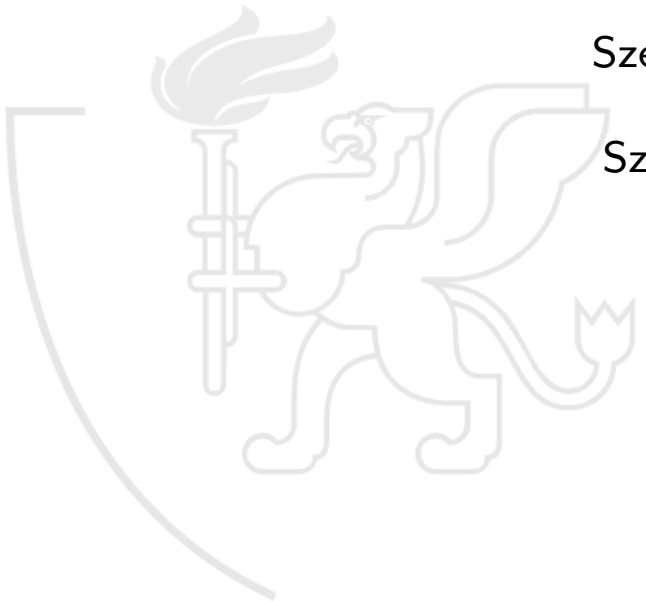
Programozás Alapjai

Dr. Gergely Tamás
Dr. Jász Judit

Szegedi Tudományegyetem
Informatikai Intézet
Szoftverfejlesztés Tanszék



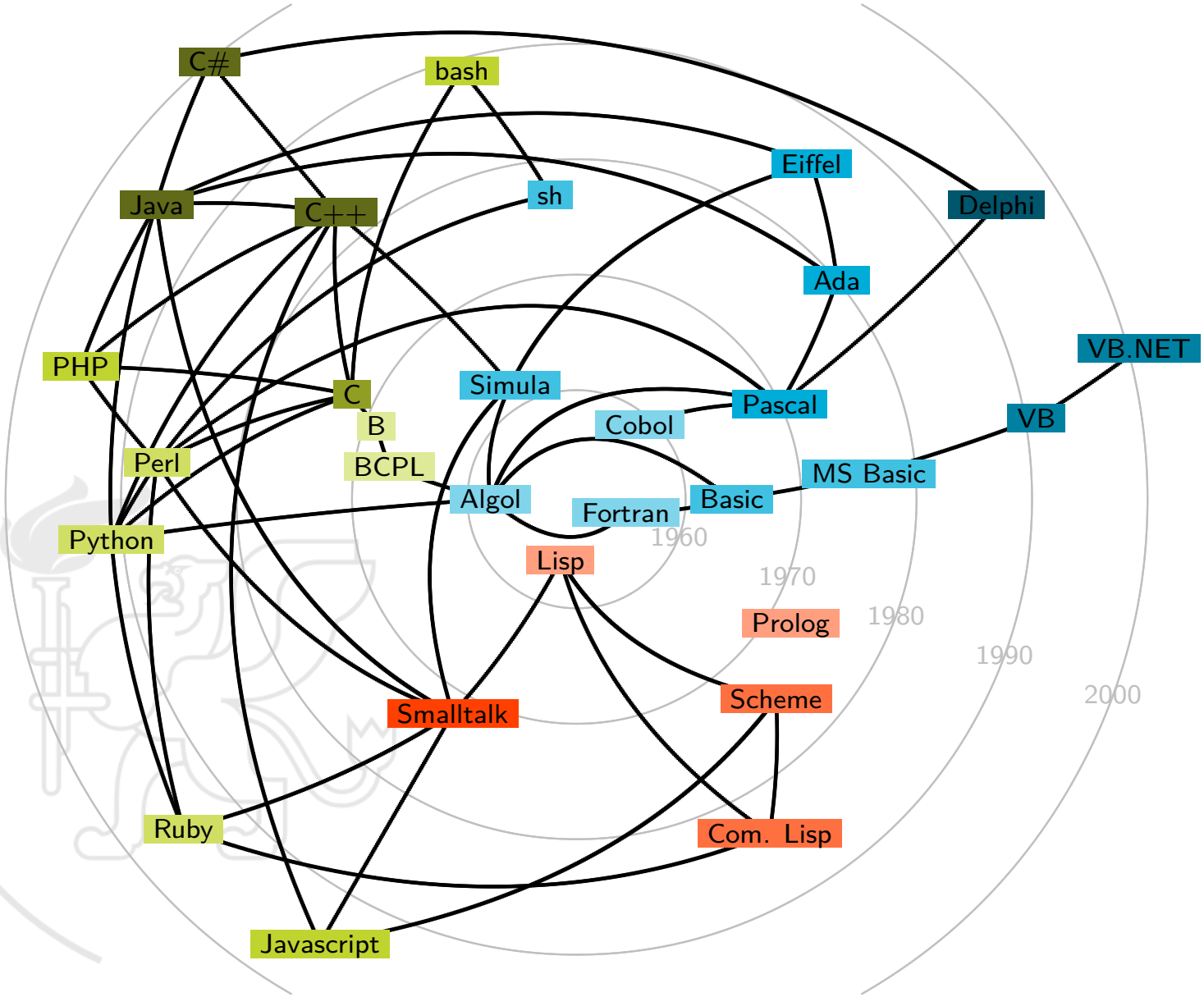
2020

(v0908)

Tartalom

Tartalom

Programozási nyelvek/paradigmák



Programozási alapfogalmak

Programozás *Valós problémák számítógépes megoldása.*

- Számítógép tulajdonságai
  - Általános célú (univerzális)
    - A határok egyre inkább elmosódnak (pl. androidos mobil, set-top-box, digitális tv, dvd/bd lejátszók).
  - Automatikus vezérlésű
    - A programját külső beavatkozás nélkül képes végrehajtani.
  - Elektronikus
    - Az áramköri kapcsolás nem mechanikus.
  - Digitális
    - Két értékből (0,1) felépített jól elkülöníthető állapotok.

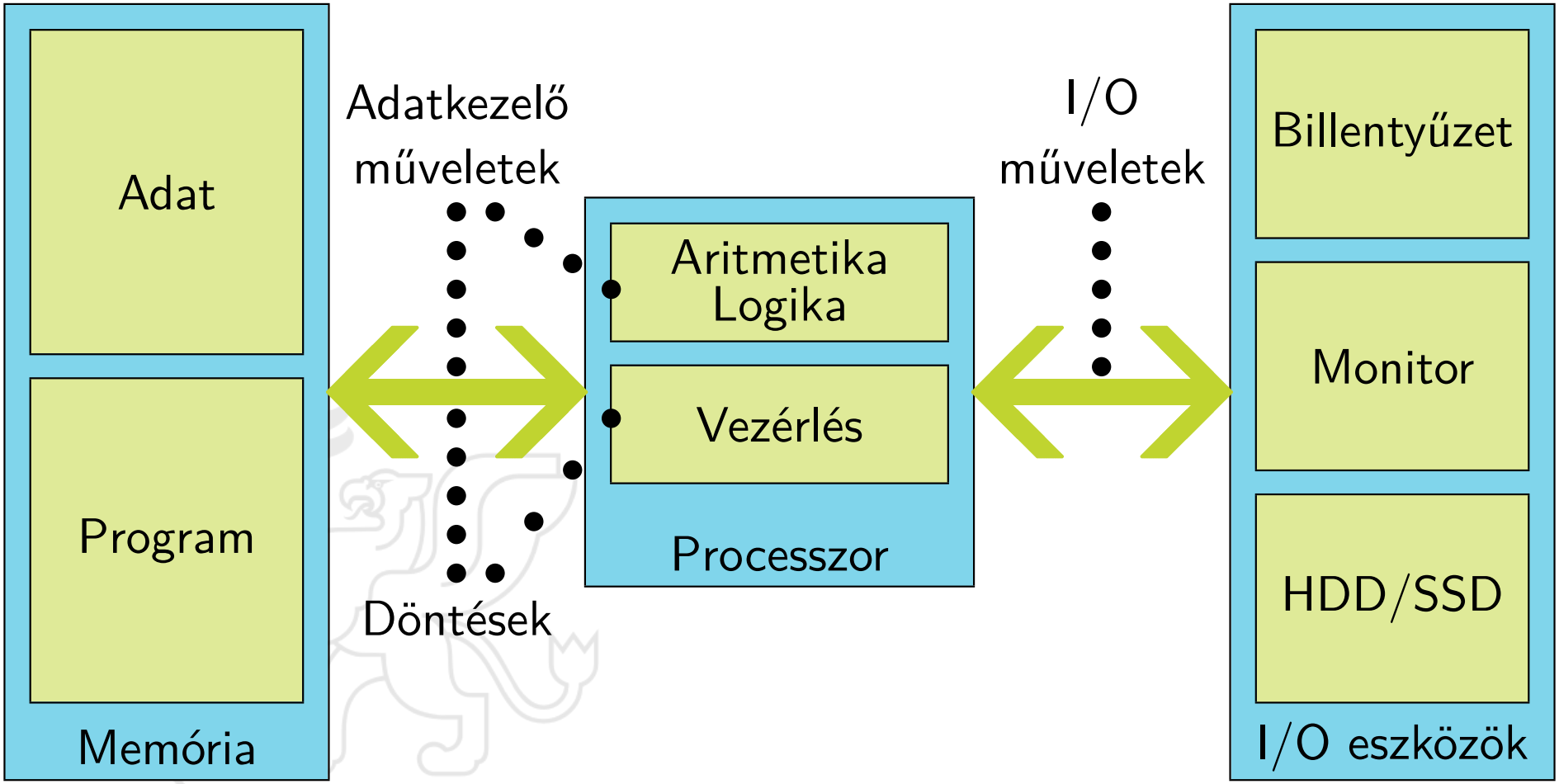
A számítógép modellje

Az ember mint modell

- Hogyan számol az ember?
  - Olvassa az utasításokat.
  - Értelmezi.
  - Végrehajtja.
  - Rátér a következő utasításra, vagy arra, amit előírnak.
- Milyen utasítások vannak?
  - Bemeneti/kimeneti (input/output)
  - Adatkezelési
  - Aritmetikai
  - Döntési

A számítógép modellje

Egyszerűsített Neumann-architektúra



A számítógép modellje

Nyelv

- A számolási tevékenységet végző ember számára magyar nyelven lehet olyan utasításokat adni, amit képes végrehajtani.
- A számítógép számára a vezérlő egység által értelmezhető és végrehajtható utasítások (parancsok) adhatók.
- Kezdetben az ember megtanulta ezeket az utasításokat (a számítógép nyelvét). A kommunikáció ezen a nyelven lassú, nehézkes, sok hibával jár.
- Miért nem tanul meg inkább a számítógép magyarul?
  - Nyelvi többértelműség. (Miért nyúl a nyúl?)
  - Szövegkörnyezettől való függőség. (A szomszédba csapott a villám, de én fogtam a nyelét.)
  - Szemantika. (El kellene magyarázni a szavak jelentését.)
- Megoldás: *Az ember is és a számítógép is tanuljon meg egy nyelvet!*

A számítógép modellje

Algoritmus

- A nyelv egy kifejező eszköz valaminek a leírására. De mit akarunk a számítógép számára leírni?
- Azt, hogy mit csináljon, vagyis milyen tevékenységet hajtson végre.
- Egy számítógép csak azt tudja megcsinálni, amit megmondunk neki, de amiről meg tudjuk mondani, hogy hogyan kell csinálni, azt a számítógép meg tudja csinálni!
- Alapvetően azt kell megmondani a számítógépnek, hogy milyen adatokon milyen műveleteket kell elvégezni. Ezt pontosan meg kell fogalmazni, vagyis el kell készíteni a problémát megoldó algoritmust.

Algoritmus

Algoritmus *Adott típusú összes feladat megoldására vonatkozó pontos előírás, amely megmondja, hogy a kezdeti adatokon milyen műveleteket milyen sorrendben kell elvégezni.*

- Az algoritmus tulajdonságai
  - Meghatározott.
    - Véges módon leírható, ugyanazon bemeneti adatokra mindig ugyanazt eredményezi.
  - Széleskörű.
    - Egész feladatosztályra vonatkozik.
  - Véges.
    - Véges számú lépésben véget ér.
  - Nem árt, ha potenciálisan megvalósítható.
- Érdekesség:
  - Nincs olyan algoritmus, amelyik el tudja dönteni, hogy egy tetszőleges program a kezdőadatokkal végrehajtva véges lépés után megáll-e.

Fogalmak

Programozási nyelv	*Olyan nyelv (szintaktikai és szemantikai szabályok összessége), amely egy problémát megoldó algoritmus leírására szolgál.*

- Programozási nyelv lehet ...
  - ... az emberi nyelvhez közel álló (magas szintű)
  - ... a számítógép nyelvéhez közel álló (alacsony szintű)

Program	*Egy algoritmusnak egy adott programozási nyelven történő leírása.*
Hardver	*A számítást végző fizikai-technikai rendszer.*
Szoftver	*A hardvert működtető programok és parancsok összessége.*
Adat	*A hardver és a szoftver által feldolgozott információ.*

Hardver környezet

- Nagyon sokféle hardver létezik, de az alapfelépítése mindnek nagyon hasonló.
  - Központi feldolgozó egység (CPU, Processzor)
    - Jellemzők: felépítés (CISC, RISC), sebesség (MHz, GHz), magok száma
  - Memória
    - Jellemzők: típus (ROM, RAM), méret (bit, byte, kB, KiB, MB, MiB, GB, GiB)
  - Háttértár (HDD, SSD, CD, DVD, BD, Flash)
    - Jellemzők: méret (MB, MiB, GB, GiB, TB, TiB)
  - Felhasználói terminál és perifériák
    - Billentyűzet, egér, képernyő, hanggenerátor, nyomtató, modem, hálózati csatoló, speciális eszközök

Szoftver környezet

- Egy hierarchikus felépítésű programrendszer.
- Főbb szintjei felhasználó szemszögből:
  - Felhasználói programok
  - Programfejlesztői rendszerek
  - Operációs rendszer
  - Gépi alapszoftver
- Minden szint elemei előállíthatók programfejleszői rendszerek segítségével

Szoftver környezet

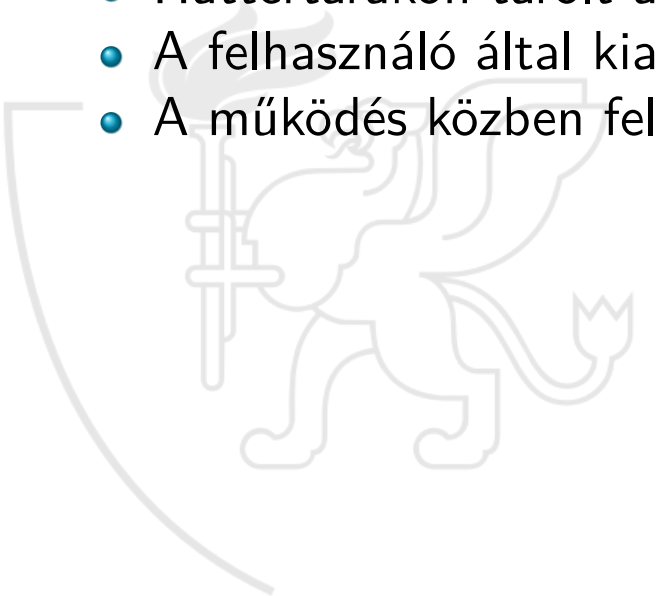
Gépi alapszoftver (BIOS)

- A hardver alapvető működését biztosító, általában ROM-ba (EPROM, EEPROM) égetett szoftver (Basic Input Output System).
- Főbb funkciói:
  - A hardver tesztelése
  - Az operációs rendszer betöltése és indítása
  - Gépi szintű be- és kimenet megvalósítása (ezt a feladatot a modern operációs rendszerek az elindításuk után részben vagy teljesen átveszik)

Szoftver környezet

Operációs rendszer

- Olyan programrendszer, amely közvetítő szerepet tölt be a számítógép hardver erőforrásai és a felhasználó között.
- Főbb funkciói:
  - Programok betöltése és végrehajtatása
  - Erőforrások elosztása
  - Input/output műveletek végzése
  - Háttértárakon tárolt adatrendszerek kezelése
  - A felhasználó által kiadott parancsok értelmezése és végrehajtása
  - A működés közben fellépett hibák lekezelése



Szoftver környezet

Programfejlesztő rendszer

- Feladata, hogy támogassa a programozás során a programok (programrendszerek) létrehozását, módosítását és végrehajtását.
- Főbb funkciói:
  - Könyvtárkezelés
  - Szövegszerkesztés
  - Fordítás
  - Végrehajtás
  - Hibakeresés
  - Rendszerparaméterezés
  - Tudakozódás

Szoftver környezet

Felhasználói program(rendszer)

- A megoldandó probléma számítógépes megoldását szolgáltatja.

Tartalom

A programozás fázisai

Szoftverfejlesztési modellek

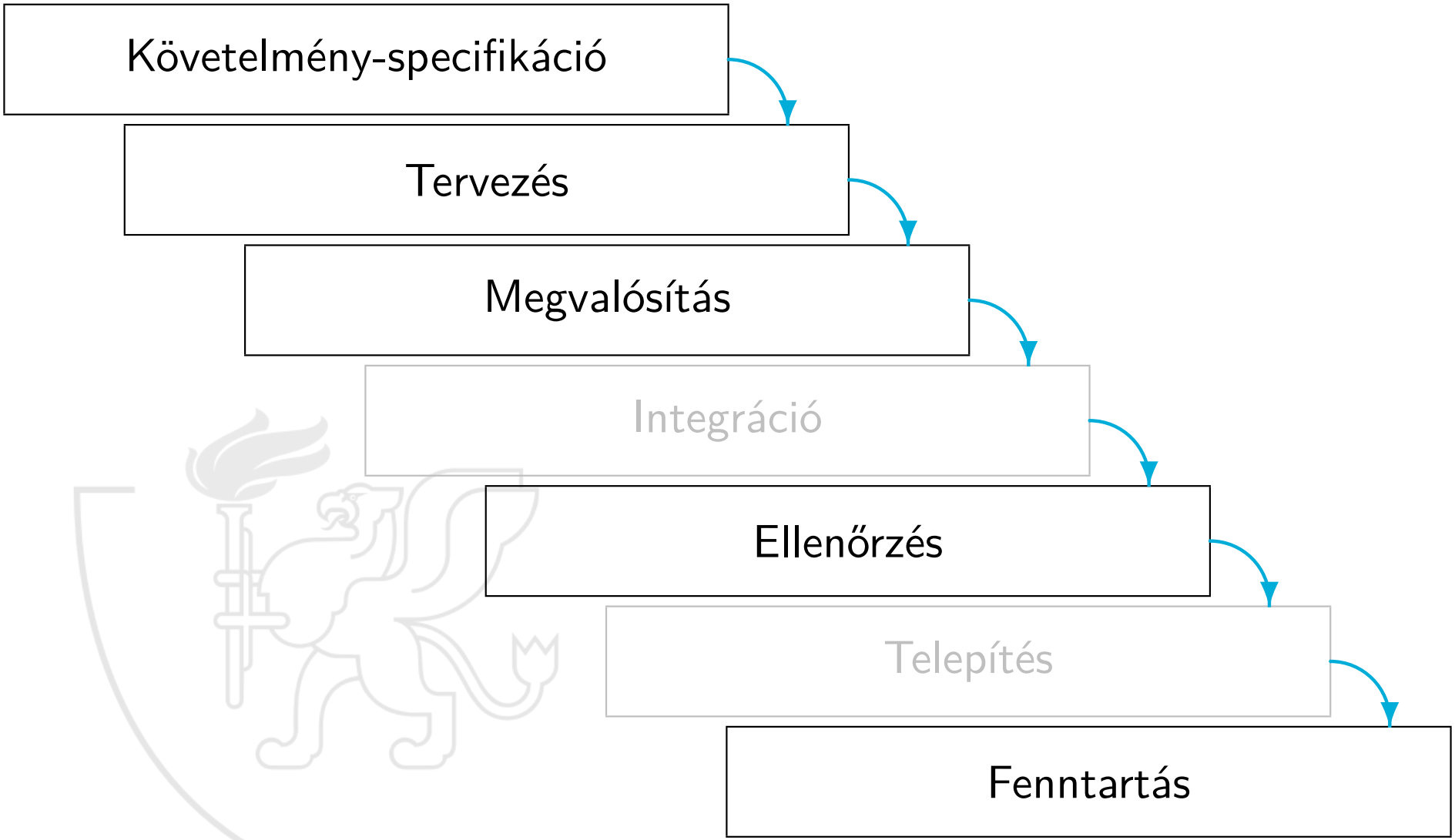
- A számítógépes problémamegoldás (a programozás) egymástól jól elkülöníthető fázisokból épül fel, amelyek sajátos kölcsönhatásban vannak egymással; ezt a kapcsolatot fejezi ki a *szoftverfejlesztési modell*.
- A világon többféle modell létezik, ezek közül nincs jó vagy rossz, csak adott feladathoz jobban vagy kevésbé alkalmas.
  - Vízesés-modell
  - V-modell
  - spirál-modell
  - O-modell
  - extrém programozás
  - ...

A programozás fázisai

Vízesés modell

- Az egyik első és legegyszerűbb a vízesés (waterfall) modell.
  - Követelmény-specifikáció (Problémafelvetés, Specifikáció)
  - Tervezés (Algoritmustervezés)
  - Megvalósítás
  - Integráció
  - Ellenőrzés (Helyességigazolás, Költségelemzés, Tesztelés)
  - Telepítés
  - Fenntartás (Végrehajtás, Fenntartás)
- Az modellben az előbbi fázisokat sorban egymás után hajtjuk végre.

A vízesés modell



A vízesés modell fázisai

Követelmény-specifikáció

- A (későbbi) felhasználó a saját szakterületének megfelelő nyelven meg tudja fogalmazni a megoldandó problémát (*problémafelvetés*), ez viszont az algoritmus elkészítéséhez nem elég.
- Az algoritmushoz pontosan meg kell határozni, hogy milyen feltételt elégítenek ki a probléma bemenő adatai, és hogy adott bemenő adatok esetén milyen feltételt kell kielégíteni a kiszámított adatoknak (*specifikáció*).
- A specifikáció tehát egy (B, K) bemeneti-kimeneti feltétel pár: akkor fogadjuk el az algoritmust a probléma megoldásának,
  - ha valahányszor a B feltétel teljesül a bemenő adatokra,
  - mindannyiszor a K feltétel teljesül a kiszámított adatokra.
- A specifikáció meghatározásánál szinte teljes mértékben szabad kezünk van, feltéve, hogy a „*mi*” nem csak a programozót, hanem a program elkészítésében érdekelt összes felet jelenti.

A vízesés modell fázisai

Tervezés

- Itt történik a specifikációt kielégítő algoritmus létrehozása (*algoritmustervezés*).
- A felülről lefelé haladó (top-down) módszer lényege:
  - A kiindulási P problémát $P_1, \ldots, P_n$ részproblémákra bontjuk.
  - Minden P_i problémának megadjuk valamely M_i megoldását.
  - Az M_i műveleteket alkalmas módon összetéve a P problémát megoldó algoritmushoz jutunk.
  - Ezt (rekurzívan) addig folytatjuk, amíg a választott programozási nyelv elemi műveletével közvetlenül meg nem adható problémákhoz jutunk.
- Az algoritmus meghatározásában mindaddig szabad kezünk van, amíg az a megadott specifikációnak megfelelően működik (figyelembe véve, hogy a specifikációban **nem** szereplő funkciókat a programnak **nem** kell és **nem** is szabad ellátnia).

A vízesés modell fázisai

Megvalósítás

- Megvalósításon az algoritmustervezés során kifejlesztett algoritmusnak egy adott programozási nyelven történő leírását értjük (*kódolás*).
- A lépés eredménye a program.
- A megvalósítás során szigorúan követnünk kell a tervezés során megalkotott algoritmust; a lehetséges variánsok, alternatívák csupán technikai jellegűek.

A vízesés modell fázisai

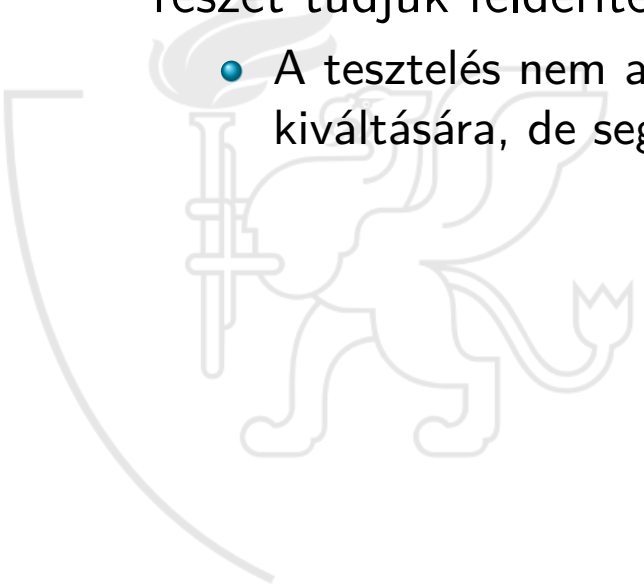
Integráció

- Amennyiben az általunk megoldott probléma egy nagyobb, összetettebb probléma része, úgy a programunknak is szükségképpen illeszkednie kell egy a nagyobb probléma egyéb komponenseinek megvalósításához.
- Ilyen esetekben ez a lépés szolgálja a saját kódunk nagyobb programrendszerbe történő beillesztését, és a beillesztés során esetleg felvetődő problémák megoldását.

A vízesés modell fázisai

Ellenőrzés

- Az ellenőrzésnek része a
  - *helyességigazolás*, minek során megmutatjuk, hogy a kifejlesztett algoritmus valóban a kiindulási probléma megoldását adja (nincs tervezési hiba);
  - *költségelemzés*, ami az algoritmus idő- és tárigényének (költségének) a bemenő adatok függvényében történő meghatározása;
  - *tesztelés*, amivel a program megvalósítása során elkövetett hibák egy részét tudjuk felderíteni.
    - A tesztelés nem alkalmas a helyességigazolás és költségelemzés kiváltására, de segíthet ezek elvégzésében.



A vízesés modell fázisai

Telepítés

- Amennyiben szükséges a programot telepíteni kell, azaz el kell helyezni és működőképessé kell tenni a felhasználó által kívánt környezetben.

A vízesés modell fázisai

Fenntartás

- A programkészítés végső célja az, hogy a megoldandó probléma konkrét bemenő adataira végrehajtsuk a kifejlesztett algoritmust.
- A program fenntartásán a végrehajtás során felmerült problémák megoldását értjük, ide tartozik például:
  - a használat során felmerülő hibák kijavítása, vagy
  - apróbb módosítások elvégzése.

Dokumentáció

- Nagyon fontos, hogy a teljes folyamat dokumentálva legyen, és ne csak ott, ahol ez nyilvánvalóan szükséges.
  - A specifikáció önmagában dokumentum.
  - A tervek valamilyen formájú leírása.
  - Megjegyzések a kódban.
  - Integráció során történt módosítások leírása.
  - Számítások eredménye, teszt-jegyzőkönyv.
  - Telepítési kézikönyv.
  - Felhasználói kézikönyv, hibajavítások, módosítások rögzítése.

Tartalom

A programok fő komponensei

- Minden programnak két fő komponense van:
  - az adatok, és
  - az adatokon végzett műveletek.
- Az adatok is és a műveletek is lehetnek elemiek és összetettek.
- Adatokból adat-összetételi, műveletekből műveletképzési szabályokkal összetett adatok illetve műveletek gyárthatók.

Az algoritmus vezérlése

Algoritmus vezérlése *Az az előírás, amely az algoritmus minden lépésére (részműveletére) kijelöli, hogy a lépés végrehajtása után melyik lépés végrehajtásával folytatódjék (esetleg fejeződjék be) az algoritmus végrehajtása.*

- Az algoritmusnak, mint műveletnek a vezérlés a legfontosabb komponense.
- Ezt az előírást nevezzük az algoritmus vezérlésének.

Vezérlési módok

- A vezérlési mód azt fejezi ki, hogy egyszerűbb műveletekből hogyan építünk fel összetett műveletet és ennek milyen lesz a vezérlése.
- Négy fő vezérlési módot különböztetünk meg:

Szekvenciális Véges sok adott művelet rögzített sorrendben egymás után történő végrehajtása.

Szelekciós Véges sok rögzített művelet közül adott feltétel alapján valamelyik végrehajtása.

Ismétléses Adott művelet adott feltétel szerinti ismételt végrehajtása.

Eljárás Adott művelet alkalmazása adott argumentumokra, ami az argumentumok értékének pontosan meghatározott változását eredményezi.

Vezérlési módok és C

- Mint az látható, a C megismert (áttekintett) művelet-összetételi konstrukciói besorolhatók a négy vezérlési mód valamelyikébe.
- A vezérlési módok ugyanakkor nyelvek feletti fogalmak.
- A imperatív (algoritmikus) programozási nyelvekben ezek a vezérlési szerkezetek (közvetlenül vagy közvetve) megvalósíthatók.

Tartalom

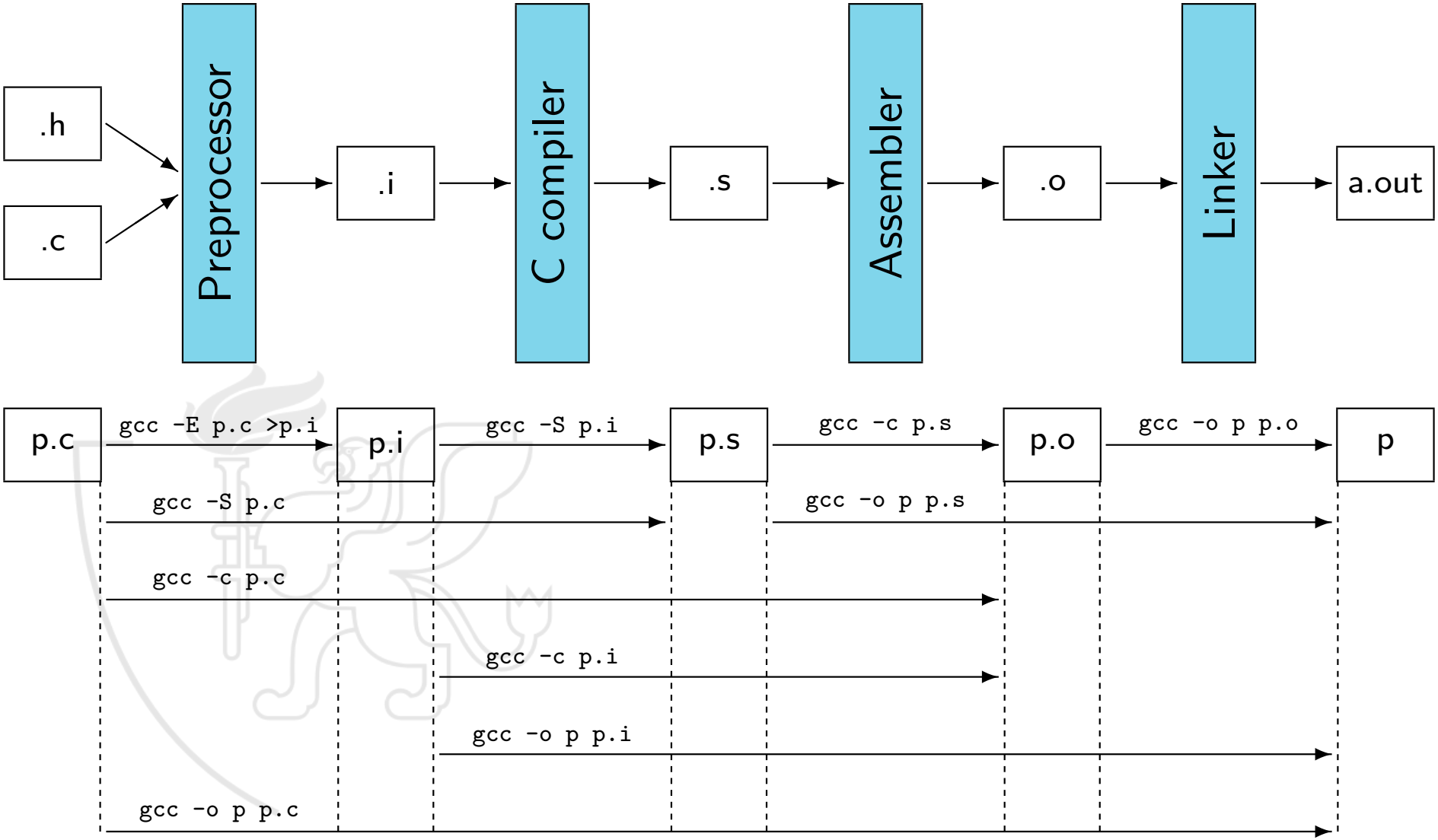
A C forrás fordításának folyamata

- 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 fordítás közben többféle fájllal dolgozunk. A szabványos végződések:

  `file.c` C source file – C forrásfájl
  `file.h` C header file – C fejléc fájl
  `file.i` preprocessed C file – előfeldolgozott C fájl
  `file.s` assembly source file – assembly nyelvű forrásfájl
  `file.o` object file – gépi kódú fájl
  `a.out` linked executable – szerkesztett futtatható fájl
- A fájl végződése utal a programozási nyelvre és arra, hogy mit kell vele csinálni.

A C forrás fordításának folyamata

Egyszerűbb programok esetén



Egy C program felépítése

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

```c
typedef unsigned long int ui32;
```

  - Változót

```c
int magassag;
```

  - Függvényt

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

  - Konstanst

```c
#define N 42 /* Ez egyben definíció is. */
```

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.)

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.

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.

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).

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.

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).

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.)

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);
```

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).

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);
```

Tartalom

Szintaxis és szemantika

- 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.
- A nyelvet a szintaxis és a szemantika együtt határozza meg.

Szintaxis	*Formai szabályok olyan rendszerét, amely meghatározza, hogy egy adott kommunikációs nyelvben melyek a szabályos jelsorozatok, a nyelv szintaxisának nevezzük.*
Szemantika	*Egy nyelv szemantikája határozza meg, hogy a szintaktikusan helyes jelsorozatok mit jelentenek.*

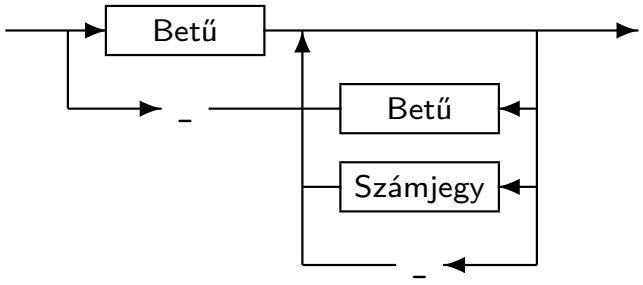
Szintaxisdiagram

- A szintaxis megadására számos módszer ismeretes, mi a kurzuson szintaxis diagramokat használunk.
- Ebben minden szintaktikus egység egyedi elnevezést kap, és a szintaktikus egységhez tartozó szabályos jelsorozatokat egy diagram (ábra) definiálja.
- Az ábrán a szintaktikus egységneveket tartalmazó dobozokat (téglalapokat) és konkrét jelsorozatokat irányított vonalak kötik össze.
- A diagramban a konkrét jelsorozatok önmagukat definiálják.
- Minden diagramnak egy bemenete (kezdőpontja) és egy kimenete (végpontja) van.
- Szintaxis diagramok rendszerében egy diagram azokat és csak azokat a jelsorozatokat határozza meg, amelyek úgy kaphatók, hogy a diagram bemenetéről indulva az irányított vonalak mentén a kijáratig haladva valahányszor érintünk egy egységet, leírjuk az általa meghatározott jelsorozatok egy elemét.

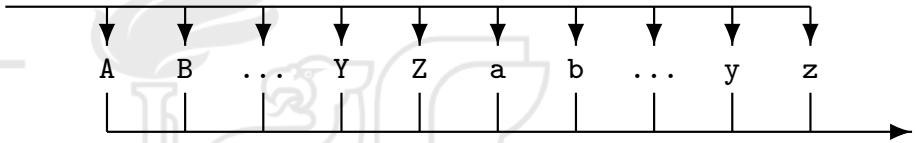
Szintaxis

Azonosító, betű, számjegy

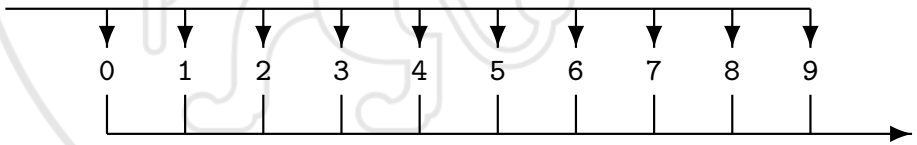
- Azonosító



- Betű



- Számjegy



Tartalom

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.

A `bool` adattípus

- A C99 szabvány óta lehet használni a `_Bool`, 0 és 1 értékekkel rendelkező adattípust, ez azóta a nyelv része.
- A `bool` adattípus a C nyelvnek nem eleve definiált adattípusa, hanem a C99 szabvány óta egy hivatalos standard kiegészítése.
- A `bool` típus értékkészlete a `true` és `false` konstans értékekből áll.
- A `bool` típus használatához a program elejére be kell szúrni az alábbi sort:

```
#include <stdbool.h>
```

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 értékek műveletei

Relációs műveletek

- `_Bool`×`_Bool`→`_Bool`
  (*logikai*×*logikai*→*logikai*)
  két operandusú műveletek

  `==` *azonosság*

  `!=` *kizáró vagy*

  `<` inv. imp. tag.
  $\neg(a \leftarrow b)$

  `>` imp. tag.
  $\neg(a \rightarrow b)$

  `<=` *implikáció*
  $a \rightarrow b$

  `>=` *inverz implikáció*
  $a \leftarrow b$

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

Az `int` adattípus

- A C nyelv egész értékek tárolására alkalmas eleve definiált elemi adattípusa.
- Értékkészlete az [**INT_MIN**, **INT_MAX**] zárt intervallumba eső egész számok halmaza.
- Az `INT_MIN` és `INT_MAX` eleve definiált konstans azonosítók, használatukhoz a program elejére be kell szúrni az alábbi sort:

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

Az `int` (egész) adattípus műveletei

Aritmetikai műveletek

- `int`→`int`
  (*egész*→*egész*)
  egy operandusú műveletek
  - `-` előjelváltás
- `int`×`int`→`int`
  (*egész*×*egész*→*egész*)
  két operandusú műveletek
  - `+` összeadás
  - `-` kivonás
  - `*` szorzás
  - `/` egészosztás
  - `%` maradékképzés

```
-a
 a + 8
 7 - a
 6 * 7
 a / 3
42 % a
```

Az `int` (egész) adattípus műveletei

Relációs műveletek

- `int`×`int`→`_Bool`
  (*egész*×*egész*→*logikai*)
  két operandusú műveletek

`==`	egyenlő
`!=`	nem egyenlő
`<`	kisebb
`>`	nagyobb
`<=`	kisebb vagy egyenlő
`>=`	nagyobb vagy egyenlő

```
3  ==  a
a  !=  7
a  <   42
a  >   77
a  <=  b
8  >=  a
```

Az `int` adattípus műveleteinek tulajdonságai

- 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
```

A `double` adattípus

- A C nyelv valós értékek tárolására alkalmas eleve definiált elemi adattípusa (a matematikai valós számok és műveleteik számítógépes modellezésére használható).
- Értékkészlete a $[-(2-2^{-52})\cdot 2^{1023}, (2-2^{-52})\cdot 2^{1023}]$ zárt intervallumba eső valós számok részhalmaza úgy, hogy ebből az intervallumból minden valós szám egy adott e relatív pontossággal megközelíthető egy `double` adattípusbeli értékkel.
- Ez azt jelenti, hogy bármely a valós számhoz van olyan `x` `double` típusú érték, hogy

$$\left|\frac{\mathtt{x}-a}{a}\right| \leq e.$$

- A közelítés mértéke a típus tárolt értékeinél adott, de számítás közben az adott architektúrán való megvalósítás függvényében ettől eltérhet.

A `double` (valós) adattípus műveletei

Aritmetikai műveletek

- `double`→`double`
  (*valós*→*valós*)
  egy operandusú műveletek
  - `-` előjelváltás
- `double`×`double`→`double`
  (*valós*×*valós*→*valós*)
  két operandusú műveletek
  - `+` összeadás
  - `-` kivonás
  - `*` szorzás
  - `/` osztás

```
-a
 a   + 8.0
 7.0 - a
 6.0 * 7.0
 a   / 3.0
```

A `double` (valós) adattípus műveletei

Relációs műveletek

- `double`×`double`→`_Bool`
  (*valós*×*valós*→*logikai*)
  két operandusú műveletek

  `==` egyenlő (óvatosan!)

  `!=` nem egyenlő (óvatosan!)

  `<` kisebb

  `>` nagyobb

  `<=` kisebb vagy egyenlő

  `>=` nagyobb vagy egyenlő

```
 3.0 == a
   a != 7.2
   a <  1e-10
   a >  0.2E3
   a <= b
42.0 >= a
```

Az double adattípus műveleteinek tulajdonságai

- 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
```

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

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

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

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

```
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
```

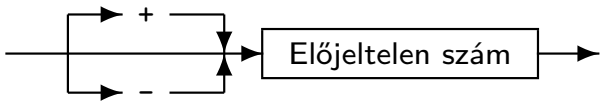
Numerikus adattípusok

- Az `int`, `float` és `double` adattípusokat összefoglalóan numerikus adattípusoknak nevezzük.
- A numerikus adattípusok értékeinek leírására számleírást, röviden számokat használunk.
  - Az egész számokat 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.) A `gcc` ismeri a bináris számleírást is, de ez nem szabvány!
  - Valós számok leírására tizedes törtet használhatunk, tízes exponenssel kiegészítve.

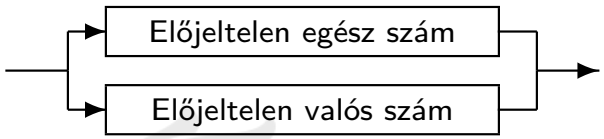
Szintaxis

Szám, előjeltelen szám

- Szám



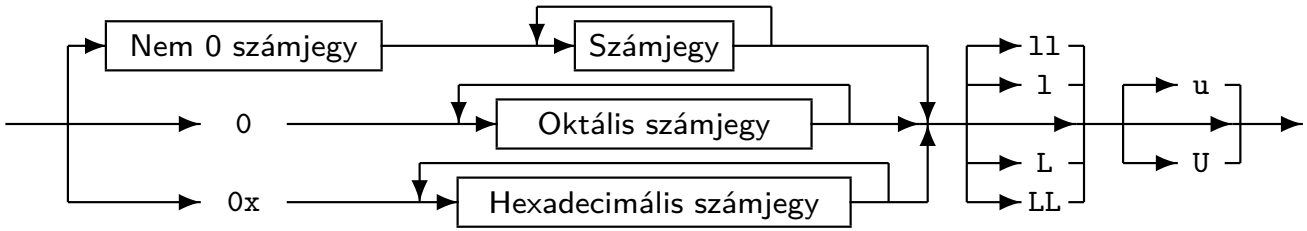
- Előjeltelen szám



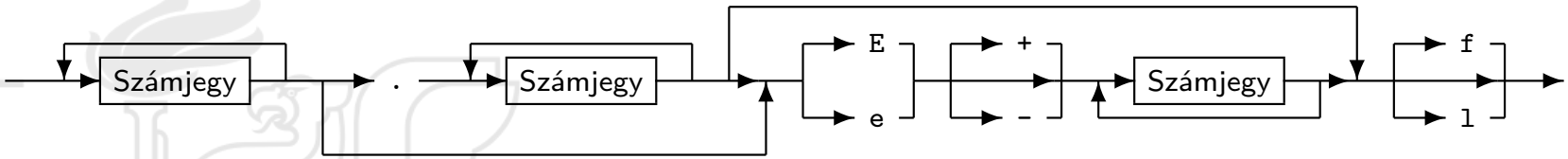
Szintaxis

Előjeltelen egész és előjeltelen valós szám

- Előjeltelen egész szám



- Előjeltelen valós szám



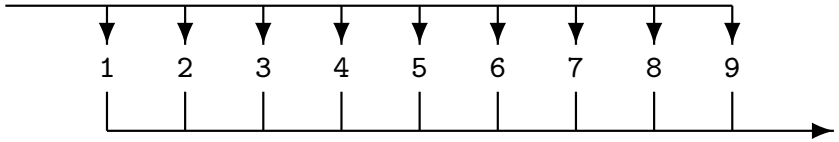

```
42
052
0x2a
0b101010 /* gcc! */
```

```
-3.1415
3e5
3E+6
271.72e-2
```

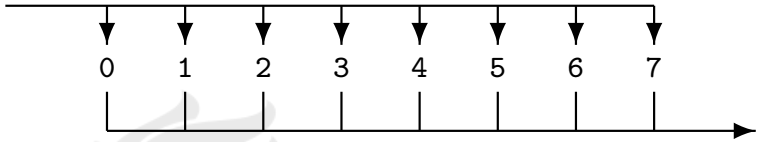
Szintaxis

Nem nulla, oktális és hexadecimális számjegyek

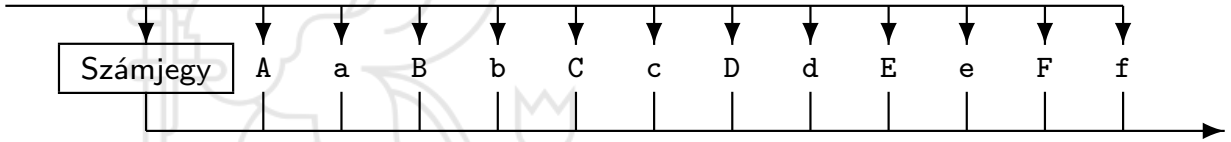
- Nem 0 számjegy



- Oktális számjegy

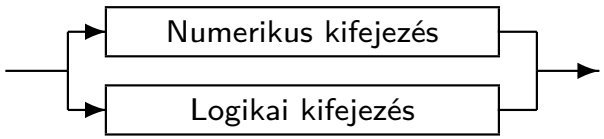


- Hexadecimális számjegy



Kifejezés

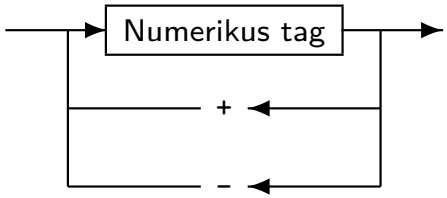
- Kifejezésen olyan programkomponenst értünk, amely egy adattípus értékének olyan jelölése, amely műveleteket is tartalmazhat.
- A kifejezés által jelölt értéket a kifejezés kiértékelése határozza meg.
- A kifejezés szintaxisa (egyelőre)



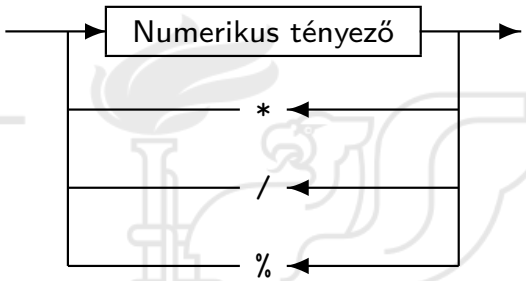
Szintaxis

Numerikus kifejezés és tag

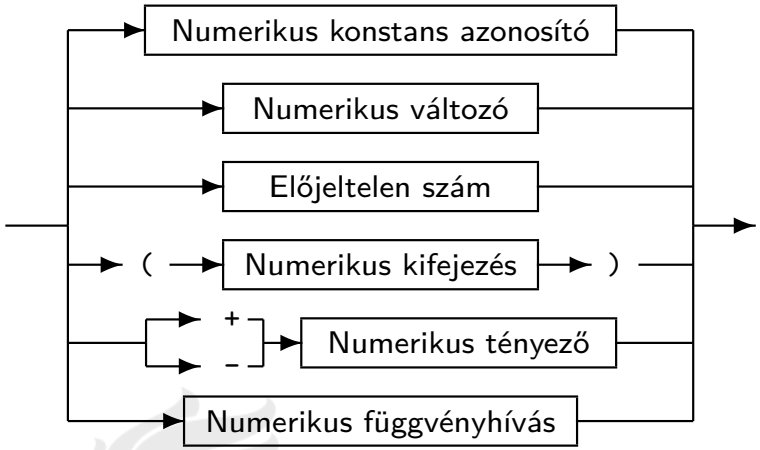
- Numerikus kifejezés



- Numerikus tag



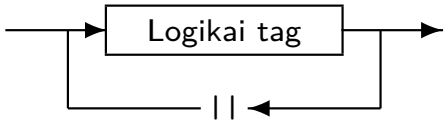
- Numerikus tényező



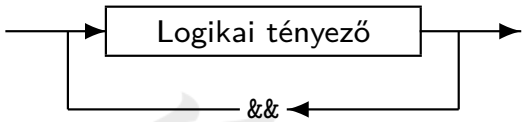
Szintaxis

Logikai kifejezés és tag

- Logikai kifejezés

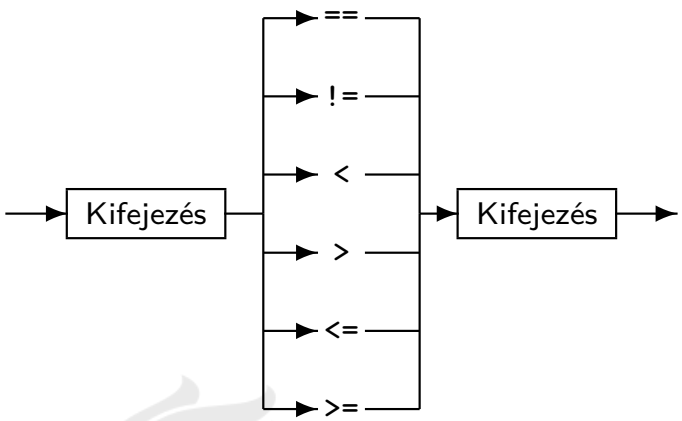


- Logikai tag



- Logikai tényező

- Relációs kifejezés



Kifejezések kiértékelése

- A kifejezés kiértékelését két előírás együttesen határozza meg:
  - a kifejezés szerkezetén alapuló prioritási előírás, és
  - a haladás irányára vonatkozó asszociativitási előírás.

Prioritási előírás

- A műveletek között definiálva van egy erősségi sorrend, az úgynevezett prioritás.
- Ez azt jelenti, hogy egy $A \odot_1 B \odot_2 C$ alakú kifejezés, amelyben $\odot_2$ magasabb prioritású művelet, mint $\odot_1$, az $A \odot_1 (B \odot_2 C)$ zárójelezésnek megfelelően értékelődik ki.

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

Műveletek asszociativitása

- Azonos prioritású műveletek esetén a kiértékelést az asszociativitás iránya szerint kell elvégezni.
- Ez azt jelenti, hogy egy $A \odot_1 B \odot_2 C$ alakú kifejezés, amelyben $\odot_1$ és $\odot_2$ azonos prioritású műveletek,
  - balról jobbra asszociativitás esetén $(A \odot_1 B) \odot_2 C$,
  - jobbról balra asszociativitás esetén pedig $A \odot_1 (B \odot_2 C)$

  zárójelezésnek megfelelően értékelődik ki.

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

C műveletek prioritása

műveletek	asszoc.	C operátorok
egyoperandusú postfix, elemkiválasztás, mezőkiválasztás, függvényhívás	→	`x++`, `x--`, `[]`, `.`, `->`, `f()`
egyoperandusú prefix, méret, típuskényszerítés	←	`+`, `-`, `++x`, `--x`, `!`, `~`, `&`, `*`, `sizeof`, `(type)`
multiplikatív	→	`*`, `/`, `%`
additív	→	`+`, `-`
bitléptetés	→	`<<`, `>>`
kisebb-nagyobb relációs	→	`<=`, `>=`, `<`, `>`
egyenlő-nem egyenlő relációs	→	`==`, `!=`
bitenkénti 'és'	→	`&`
bitenkénti 'kizáró vagy'	→	`^`
bitenkénti 'vagy'	→	`|`
logikai 'és'	→	`&&`
logikai 'vagy'	→	`||`
feltételes	←	`?:`
értékadó	←	`=`, `+=`, `-=`, `*=`, `/=`, `%=` `>>=`, `<<=`, `&=`, `^=`, `|=`
szekvencia	→	`,`

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.

Numerikus kifejezések típusa

- Minden numerikus kifejezés egész vagy valós típusú.
- A kifejezés típusának meghatározása a kifejezés felépítése szerinti indukcióval a következő:
  - A tényező típusa:
    - Ha változó, akkor a deklarációjában megadott típus;
    - Ha konstans vagy literál, akkor a számleírás szerinti típus;
    - Ha `(K)` alakú, akkor a `K` kifejezés típusa;
    - Ha `+T` alakú, akkor a `T` tényező típusa;
    - Ha `-T` alakú, akkor a `T` tényező típusa;
    - Ha függvényhívás, akkor a függvényművelet eredménytípusa.
  - A tag típusa:
    - Ha `A%B` alakú, akkor egész (és `A` és `B` is csak egész lehet);
    - Ha `A*B` vagy `A/B` alakú és `A` és `B` is egész, akkor egész;
    - Ha `A*B` vagy `A/B` alakú és `A` vagy `B` valós, akkor valós.

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
```

Műveletek számokkal

Típuskonverziók

- Az `int` (egész) adattípus nem része a `double` (valós) adattípusnak.
  - Előfordulhat pl. olyan egész érték amely nem ábrázolható pontosan valós típusúként.
- Valós típusú kifejezés akkor sem szerepelhet csak egész típuson értelmezett műveletben (pl. %), ha az értéke egész.
- Valós típuson is értelmezett műveletekben (+, −, ∗) használható viszont egész típus valós helyett.
  - Az ilyen műveletek akkor jelölnek egész típuson végzendő egész típusú értéket eredményező műveletet, ha mindkét operandusuk egész; különben valós típusú értéket eredményeznek.
  - Ilyenkor az egész típusú operandus a művelet előtt átkonvertálódik valós típusúvá, ami egy implicit (a fordító által automatikusan generált) konverziós műveletet jelent.
    - Ezért ajánlatos a `2*x` művelet helyett `2.0*x` műveletet használni, ha az `x` változó `double` 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 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));
```

Tartalom

Utasítások

- 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()`).

Összetett utasítások képzése

- Tudunk tehát kifejezésekből egyszerű utasításokat gyártani.
- Most mélyebb magyarázat nélkül átnézzük (átismételjük), hogy a C nyelv milyen lehetőségeket ad összetett utasítások képzésére, illetve megadjuk ezen konstrukciók szintaxisát.

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.

```
1 {
2     e = 54321;
3     o = e / 3600;
4     e = e % 3600;
5     p = e / 60;
6     m = e % 60;
7 }
```

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

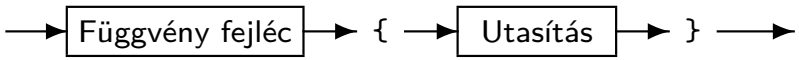
```
int c;
c = f(3,5);
```

Függvény

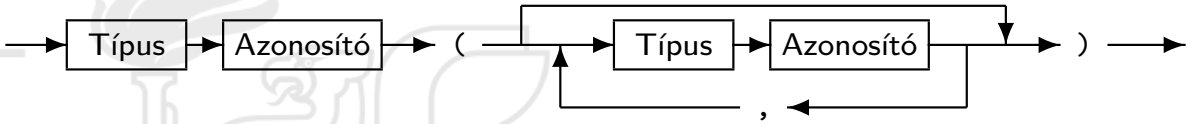
Szintaxis

- Függvény deklaráció

- Függvény definíció (egyben deklaráció is)



- Függvény fejléc



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 `return` utasítás

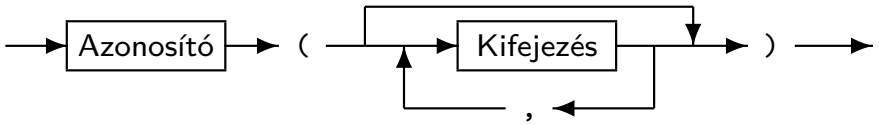
Szintaxis

- A `return` utasítás szintaxisa C-ben

Függvényhívás

Szintaxis

- Függvényhívás szintaxisa C-ben



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).

Függvények

Példa

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

```
1  int masodpercek(int ora, int perc, int masodperc)
2  {
3      return 3600 * ora + 60 * perc + masodperc;
4  }
5
6  int main()
7  {
8      int mp1, mp2;
9      mp1 = masodpercek(12, 5, 7);
10     mp2 = masodpercek(6, 45, 0);
11 }
```

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?

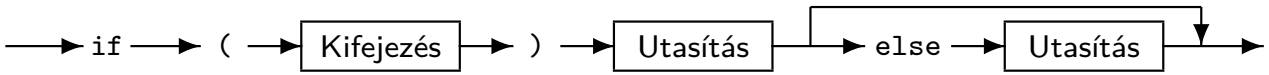
```
if(a <= b)
  k = a;
else
  k = b;
```

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

Az `if` utasítás

Szintaxis

- Az `if` utasítás szintaxisa C-ben



A `switch` utasítás

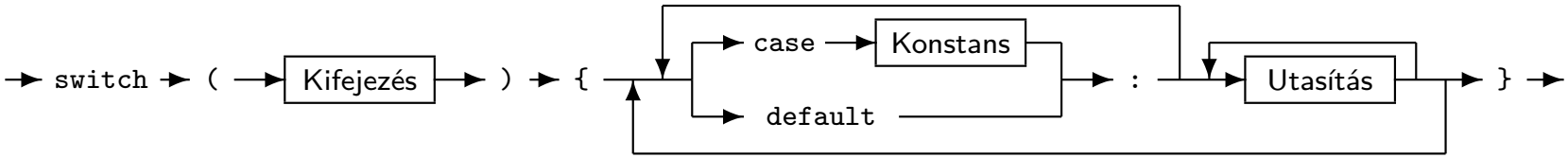
- Ha egy kifejezés értéke alapján többféle utasítás közül kell választanunk, a `switch` utasítást használhatjuk.
  - Megadhatjuk, hogy hol kezdődjön és meddig tartson az utasítás-sorozat végrehajtása.
- Gyakorlati jegyből csináljunk háromfokozatú minősítést!

```
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;
}
```

A `switch` utasítás

Szintaxis

- A `switch` utasítás szintaxisa C-ben



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 `while` utasítás

Szintaxis

- A `while` utasítás szintaxisa C-ben

A `do while` utasítás

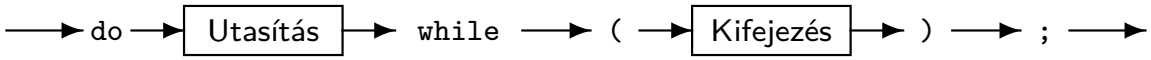
- Ha valamilyen műveletet mindaddig végre kell hajtani, amíg egy feltétel igaz a `do while` utasítás (is) használható.
  - A művelet végrehajtása szükséges a feltétel kiértékeléséhez.
  - A feltétel ellenőrzése a művelet után történik, így ha a feltétel kezdetben hamis volt, a műveletet akkor is legalább 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));
```

A `do while` utasítás

Szintaxis

- A `do while` utasítás szintaxisa C-ben



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 * x) {
    printf("%lf\n", x);
}
```

A `for` utasítás kifejezései

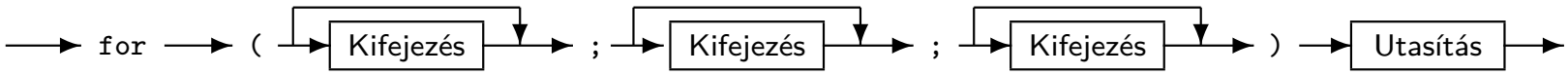
- A kif_1 (*inicializáló*) és kif_3 (*léptető*) kifejezések többnyire értékadások vagy függvényhívások, kif_2 (*feltétel*) pedig relációs kifejezés.
- A három kifejezés bármelyike elhagyható, de a pontosvesszőknek meg kell maradniuk.
  - Ha kif_1 vagy kif_3 marad el, akkor egyszerűen elmarad a kifejtésből.
  - Ha a kif_2 feltétel nem szerepel az utasításban, akkor állandóan igaznak tekintjük, így végtelen ciklust kapunk, amelyből más módon kell kiugrani (pl. `return` vagy `break` révén).

```
for (;;) { /* ... */ }
```

A `for` utasítás

Szintaxis

- A `for` utasítás szintaxisa C-ben



A `break` és `continue` utasítások

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

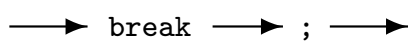
  `break` Megszakítja a ciklust, a program végrehajtása a ciklusmag utáni első utasítással folytatódik. Használható a `switch` utasításban is, hatására a program végrehajtása a `switch` utáni első utasítással folytatódik.

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

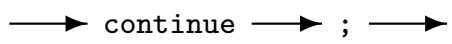
A `break` és `continue` utasítások

Szintaxis

- `break` szintaxisa C-ben



- `continue` szintaxisa C-ben



Egyszerű ki- és bevitel

- Az egyszerűbb programok be- és kiviteli utasításokkal kommunikálnak a környezetükkel.
- Ahhoz, hogy működő példát mutassunk be, szükségünk van ezekre az utasításokra is.
- A C nyelv nem tartalmaz speciális ki- és beviteli utasításokat, de ezeket megvalósító standard függvényeket, függvénykönyvtárat igen; ezek minden implementációban léteznek és ugyanúgy működnek.

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);
```