

Alkalmazott modul: Programozás

1. előadás

Szoftverfejlesztés, programozási
paradigmák

Giachetta Roberto

groberto@inf.elte.hu
http://people.inf.elte.hu/groberto

Szoftverfejlesztés

A program

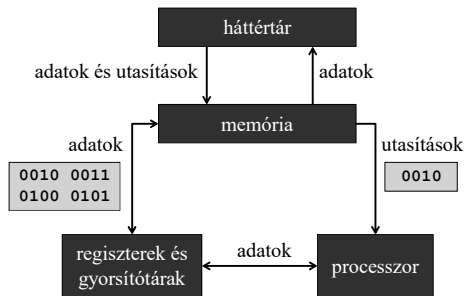
- *A program*
 - *matematikailag*: állapotterek (értékek direktszorzata) felett értelmezett reláció
 - *informatikailag*: utasítások sorozata, amelyek műveleteket hajtanak végre a megadott értékekkel, az *adatokkal*
- A programban foglalt utasítássorozatot, vagy *programkódot* a *processzor* (CPU, GPU, ...) hajtja végre
 - a processzor korlátozott utasításkészlettel rendelkezik, ezért összetett utasításokat nem képes végrehajtani
 - a végrehajtáshoz segédárakat (regiszterek, gyorsítótárak) használ, és kommunikál a *memóriával*

ELTE IK, Alkalmazott modul: Programozás

1:2

Szoftverfejlesztés

A program



ELTE IK, Alkalmazott modul: Programozás

1:3

Szoftverfejlesztés

Absztrakció a szoftverfejlesztésben

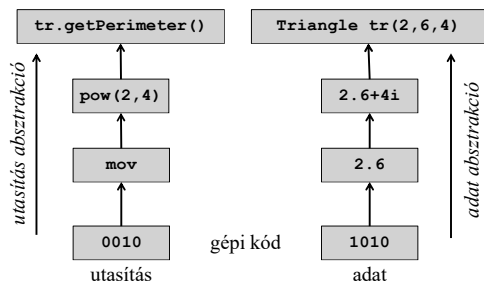
- A processzor által értelmezhető utasításkészletet és adathalmazt nevezük *gépi kódnak* (*object code*)
- A programokat ennél magasabb szinten kell elkészíteni, ezért szükségünk van a működés és az adatkezelés absztrakciójára:
 - az *utasításabsztrakció* (*control abstraction*) biztosítja, hogy a processzor egyszerű, egymást követő utasításai (összeadás, mozgatás, összehasonlítás) mellett összetett parancsokat és vezérlési módszert fogalmazzunk meg
 - az *adatabsztrakció* (*data abstraction*) lehetővé teszi, hogy különböző típusokba soroljuk adatainkat, amelyek meghatározzák az értéktartományt, és az elvégezhető műveleteket

ELTE IK, Alkalmazott modul: Programozás

1:4

Szoftverfejlesztés

Absztrakció a szoftverfejlesztésben



ELTE IK, Alkalmazott modul: Programozás

1:5

Szoftverfejlesztés

Adatok

- A adatok kétféle fajtáját tarjuk nyilván:
 - *változó* (*variable*): értéke (esetleg típusa is) változtatható a program futása során, mindig a memóriában tároljuk, és megfelelő azonosítóval (*változónév*) látjuk el
 - pl.: `a = 1` (az `a` változó értéke megváltozott 1-re)
 - a változókat *deklarálni* (név és egyéb tulajdonságait megadni), valamint *definiálni* kell (megadni a hozzájuk tartozó értéket)
 - *konstans* (*constant*): értéke és típusa rögzített, nem változhat a program futása során
 - a konstansokat is elláthatjuk azonosítóval (*nevesített konstansok*)

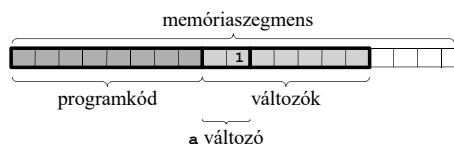
ELTE IK, Alkalmazott modul: Programozás

1:6

Szoftverfejlesztés

Adatok a memóriában

- A memória tekinthető egy byte-sorozatnak, ahol minden byte-nak sorszáma van, ez a *memóriacím*
- ehhez rendeljük az azonosítót, amivel a program futása során hivatkozhatunk rá
- minden futó program önálló memóriaterületet (*szegmenst*) kap, amelyen az ő kódja és az ő adatai helyezkednek el



ELTE IK, Alkalmazott modul: Programozás

1:7

Szoftverfejlesztés

Adatok típusa

- Az *adattípus* meghatározza a felvehető értékek halmazát (a megvalósításban a memóriában elfoglalt terület mértékét is), valamint az alkalmazható műveletek körét, vagyis

$$\text{adattípus} = \text{értékhalmoz} + \text{művelethalmoz}$$
- Minden adat rendelkezik típussal (deklarációkor adjuk meg)
- A gépi kódon értelmezhető, egyszerű típusokat nevezzük *elemi*, vagy *primitív típusoknak*:
 - logikai (\mathbb{L})
 - egész (\mathbb{Z}), természetes (\mathbb{N}) és valós (\mathbb{R}) számok
 - karakter ($\mathbb{C}\mathbb{h}$)

ELTE IK, Alkalmazott modul: Programozás

1:8

Szoftverfejlesztés

Adatok típusa

- A programozó alkothat saját, *összetett típusokat* a *típuskonstrukciók* segítségével:
 - iterált, vagy sorozat (\mathbb{D}^n , vagy \mathbb{D}^*)
 - direktsorozat ($\mathbb{D}_1 \times \mathbb{D}_2$)
 - unió ($\mathbb{D}_1 \cup \mathbb{D}_2$)
- A primitív típusokból tetszőleges bonyolultságú típust hozhatunk létre, amelyek művelethalmozát is definiálhatjuk
 - a típuskonstrukciókat halmozottan is használhatjuk
- Pl.: szöveg ($\mathbb{S} = \mathbb{C}\mathbb{h}^*$), komplex szám ($\mathbb{C} = \mathbb{R} \times \mathbb{R}$), komplex számok vektora ($\mathbb{C}^n = (\mathbb{R} \times \mathbb{R})^n$)

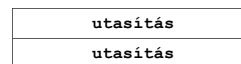
ELTE IK, Alkalmazott modul: Programozás

1:9

Szoftverfejlesztés

Utastások

- A programok építőkövei az egyszerű, processzor számára értelmezhető utastások, az úgynevezett *elemi programok*:
 - üres program: **SKIP**
 - hibás program: **ABORT**
 - értékadás: $\langle \text{változó} \rangle := \langle \text{kifejezés} \rangle$
- Az elemi programokból különböző *programkonstrukciók* segítségével hozunk létre összetett programokat:
 - szekvencia*: utastások egymásutánja



ELTE IK, Alkalmazott modul: Programozás

1:10

Szoftverfejlesztés

Utastások

- elágazás*: feltétel függvényében történő végrehajtás, amely lehet *kétágú*, vagy *többágú*



- ciklus*: utastások ismételt végrehajtása, két változata az *előtesztelő* és a *háttesztelő*



- mindkét esetben a feltétel egy logikai kifejezés

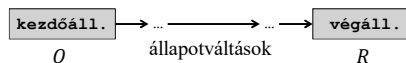
ELTE IK, Alkalmazott modul: Programozás

1:11

Szoftverfejlesztés

Állapotér, előfeltétel és utófeltétel

- Állapotérnek* (A) nevezzük a programban előforduló változók összességét (típusukat és elnevezésüket)
- Egy adott futási ponton a változók aktuális értékeinek összességét nevezzük a program *állapotának*
 - a program *állapotot vált*, ha az állapotterben lévő változók közül bármelyik értéke megváltozik
 - a program a *kezdőállapotból* indul, és a *végállapotban* terminál, ezekre külön szabhatunk logikai feltételeket, *előfeltételt* (Q), illetve *utófeltételt* (R)



ELTE IK, Alkalmazott modul: Programozás

1:12

Szoftverfejlesztés

Állapottér, előfeltétel és utófeltétel

- Pl. adjuk össze egy pozitív számokat tartalmazó vektor elemeit:

$$A = (v : \mathbb{Z}^n, s : \mathbb{Z}, i : \mathbb{Z})$$

$$Q = (\forall j \in [1..n]: (v_j = v'_j \wedge v_j \geq 1))$$

$$R = Q \wedge (s = \sum_{j=1}^n v'_j)$$

$s := 0, i := 1$
$i \leq n$
$s := s + v_i$
$i := i + 1$

ELTE IK, Alkalmazott modul: Programozás

1:13

Szoftverfejlesztés

Algoritmusok

- Algoritmus*nak nevezzük azt a műveletsorozatot, amely a feladat megoldásához vezet
 - a program lényegi része, amely nem tartalmazza az adatok beolvasását és kiírását
 - egy programban több algoritmus is szerepelhet, amelyek valamilyen kombinációja oldja meg a feladatot
- A megoldandó feladatok sokszor hasonlítanak egymásra
 - ennek köszönhetően a megoldó algoritmusok is hasonló, csupán néhány eltérést fedezhetünk fel közöttük
 - sokszor a megfelelő adatok és feltételek cseréjével megkapjuk az új feladat megoldását a korábbi alapján

ELTE IK, Alkalmazott modul: Programozás

1:14

Szoftverfejlesztés

Algoritmusok

- Az algoritmusokat ezért célszerű általánosan (absztraktn) megfogalmazni, hogy a változtatások (transzformációk) könnyen véghezvihetők legyenek
 - ha a feladathoz találunk megoldó algoritmust, akkor azt mondjuk, hogy a feladatot *visszavezettük az algoritmusra*
- Az algoritmust általában két részre szeparáljuk:
 - inicializálás*: felhasznált változók kezdőértékeinek megadása
 - feldolgozás* (mag): műveletvégzés a bemenő adatokkal és az inicializált változokkal

ELTE IK, Alkalmazott modul: Programozás

1:15

Szoftverfejlesztés

Algoritmusok

- Létező algoritmusokat azért célszerű használni, mert jó, bizonyított megoldást adják a feladatnak
 - már több ezer algoritmus létezik, amelyek mind nevesítettek
- Az egyszerű, sorozatokra alkalmazható algoritmusokat nevezzük *programozási tételek*nek, ezek a következők:
 - összegzés, számlálás
 - lineáris keresés, bináris keresés
 - maximum keresés, feltételes maximumkeresés
 - elemenkénti feldolgozás
- Számos további feladatra adható algoritmus, pl. rendezés, mintaillesztés, adattömörítés, ...

ELTE IK, Alkalmazott modul: Programozás

1:16

Szoftverfejlesztés

Algoritmusok

- Pl. a lineáris keresés (egy d adatsorozaton keressük az első olyan elemet, amely teljesíti a $\beta: \mathbb{D} \rightarrow \mathbb{L}$ feltételt, jelölje l , hogy sikerült-e megtalálni, és ind a helyét a sorozatban):

$$A = (d : \mathbb{D}^n, ind : \mathbb{Z}, l : \mathbb{L})$$

$$Q = (\forall j \in [1..n]: d_j = d'_j)$$

$$R = Q \wedge \left(l \rightarrow \left(\exists j \in [1..n]: \beta(d_j) \wedge \right. \right. \\ \left. \left. l \rightarrow (ind \in [1..n] \wedge \beta(d_{ind}) \wedge \forall j \in [1..ind-1]: \neg \beta(d_j)) \right) \right)$$

$l := \perp, i := 1$
$\neg l \wedge i \leq n$
$l := \beta(d_i), ind := i$
$i := i + 1$

ELTE IK, Alkalmazott modul: Programozás

1:17

Szoftverfejlesztés

A programozási nyelv

- Az adat-, illetve utasításabsztrakciót megvalósító eszközt nevezzük *programozási nyelv*nek
 - egy adott programozási nyelven megírt programkódot nevezünk a program *forráskódjának* (*source code*)
 - a programozási nyelv meghatározza az absztrakció szintjét, a használható típusok és utasítások halmazát
 - egy adott nyelvre rögzítettek, ám a legtöbb nyelvben a programozó által kiterjeszthetők
 - a nyelvet meghatározza a célja, vagyis milyen feladatkörre alkalmazható, továbbá a nyelv rendelkezik egy *kifejezőerővel*, azaz milyen összetett számításokat képes kifejezni

ELTE IK, Alkalmazott modul: Programozás

1:18

Szoftverfejlesztés

A programozási nyelv

- A programozási nyelvek osztályozása:
 - alacsony szintű* (assembly): a gépi kódot egyszerűsíti szövegszerűre, de nem biztosít absztrakciót, pl.:

```
data segment ; adatok
    number dw -5 ; változó létrehozása
data ends
code segment ; utasítások
...
    mov ax, number ; regiszterbe helyezése
    cmp ax, 0 ; regiszterérték összehasonlítása
    jge label1 ; ugrás, amennyiben nem negatív
    mov cx, 0
    sub cx, ax ; pozitívvá alakítás kivonással
...
```

ELTE IK, Alkalmazott modul: Programozás

1:19

Szoftverfejlesztés

A programozási nyelv

- magas szintű*: a gépi architektúrától független utasításkészlettel, nagyobb kifejező erővel rendelkezik
- lehetőséget ad az utasítás- és adatabztrakcióra
- egy egyszerű reprezentációját adja az alacsony szintű kódnak
- pl.:

```
int main() {
    int number = -5; // változó létrehozása
    if (number < 0) // ha negatív
        number = -number; // ellentettre váltás
    ...
}
```

ELTE IK, Alkalmazott modul: Programozás

1:20

Szoftverfejlesztés

Programkód átalakítás

- A programozási nyelven megírt kódot át kell alakítani a processzor számára értelmezhetővé, ennek módszerei:
 - fordítás, vagy szerkesztés (compilation)*: a kódot előzetesen lefordítjuk gépi kódra, és ezt az állományt futtatjuk
 - a program gyorsan futtatható, de gépfüggő (pl. nem indul más operációs rendszeren)
 - a műveletet a *fordítóprogram (compiler)* végzi
 - pl.: C, C++, Ada, Pascal, Eiffel, Haskell, Visual Basic
 - értelmezés (interpretation)*: a programkódot a futtatás közben alakítjuk át gépi kóddá
 - a program gépfüggetlen, de lassú a futtatása

ELTE IK, Alkalmazott modul: Programozás

1:21

Szoftverfejlesztés

Programkód átalakítás

- ezt a műveletet az *értelmező (interpreter)* végzi, amelynek a futtatás helyén rendelkezésre kell állnia
- pl.: HTML, PHP, JavaScript, Perl, Lisp, MATLAB
- futásidejű fordítás (just-in-time compilation)*:
 - a fordítás egy része előre megtörténik egy gépfüggetlen, de alacsony szintű, köztes nyelvre (*intermediate language*)
 - a futtatásakor kerül értelmezésre a köztes nyelvű kód, ezt az úgynevezett virtuális gép végzi
 - pl.: Smalltalk, Java, .NET (C#, VB.NET, ...)

ELTE IK, Alkalmazott modul: Programozás

1:22

Szoftverfejlesztés

Programkód átalakítás

- A programkód átalakítása rendszerint több lépésben történik, magasabb szintű nyelv esetén először alacsonyabb szintű kód, majd abból gépi kód készül
- A programkód tartalmazhat hibákat, amelyeket két kategóriába sorolunk:
 - szintaktikai, vagy elemzési hibák (syntax error)*: a programkód szerkezete helytelen, pl. hibás utasításnév, hivatkozás, zárójelzés, ...
 - szemantikai, vagy értelmezési hibák (semantic error)*: az érték változásával, a műveletek végrehajtásával bekövetkező hibák, pl. 0-val történő osztás, hibás memóriacím hivatkozás, ...

ELTE IK, Alkalmazott modul: Programozás

1:23

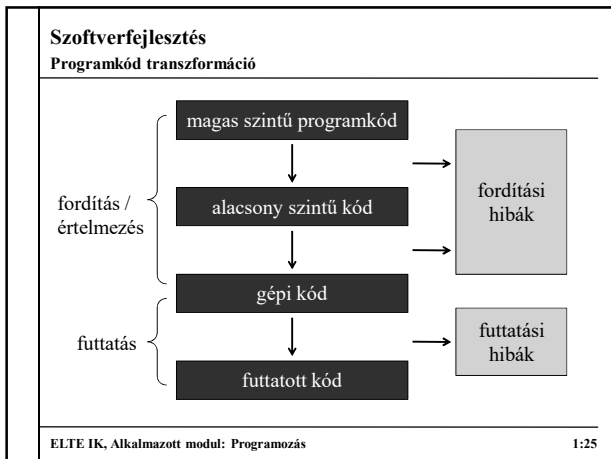
Szoftverfejlesztés

Programkód átalakítás

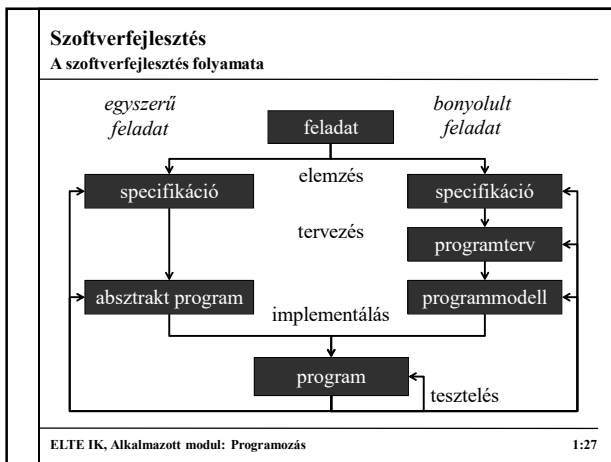
- A programhibákról az átalakítás során a lehető legpontosabb információt kapjuk (lehetséges ok és hely megadásával)
 - a fordítás során a szintaktikus hibák előre kiszűrhetők, a szemantikus hibák nagy része azonban futtatáskor léphet fel
 - értelmezés esetén csak futtatáskor szembesülünk a hibákkal
- A további programhibák ellenőrzését *teszteléssel* végezhetjük
 - a *statikus tesztelés* során a programkódot vizsgáljuk át
 - a *dinamikus tesztelés* során futás közben keressük a hibákat
- A programfejlesztői környezetek megadják a *nyomkövetés (debug)* lehetőségét is (futás közben végigkövethetjük a kódot)

ELTE IK, Alkalmazott modul: Programozás

1:24



- Szoftverfejlesztés**
A szoftverfejlesztés folyamata
- A szoftverfejlesztés a kódoláson túl több lépésből áll, amely függ a feladat bonyolultságától is:
 - A feladatot elemezni kell, és megadni a formális megfelelőjét, vagyis a *specifikációt*
 - A specifikációt alapján megtervezhető a program, amely egyszerű feladatnál az *absztrakt program*, míg bonyolult feladatnál a *programterv* elkészítésével jár, amelyből előállítható a *programmodell* (egyszerűsített célprogram)
 - A tervet implementáljuk a megfelelő programozási nyelven
 - Az implementált programot, illetve a programkódot *tesztelésnek* vetjük alá, ami módosításokat eredményezhet az implementációban (vagy a korábbi fázisokban)
- ELTE IK, Alkalmazott modul: Programozás 1:26



- Szoftverfejlesztés**
A feladat elemzése és tervezése
- Önmagában a feladat elemzése is nagyban meghatározza a programfejlesztés folyamatát, ennek két formája:
 - felülről lefelé (top-down)*: a főfeladatot részfeladatokra, majd azokat további részfeladatokra bontjuk
 - alulról felfelé (bottom-up)*: a feladatban szereplő egységeket határozzuk meg, majd azokat kombináljuk
 - A tervezés során egy nyelv-független vázát kell elkészítenünk a szoftvernek, amely megadja annak közeli működését
 - lehet formális, vagy informális modell
 - használhatunk modellező eszközt (pl. stuktogram), vagy nyelvet (pl. UML)
- ELTE IK, Alkalmazott modul: Programozás 1:28

- Szoftverfejlesztés**
Tesztelés
- A *tesztelés* annak ellenőrzése, hogy a program teljesíti-e az előírt funkcionalitást, illetve minőségi elvárásokat
 - célja elsősorban a futási idejű hibák, működési rendellenességek keresése
 - a tesztelés módja szerint lehet:
 - fekete doboz tesztelés*: a tesztelendő programrész ismeretlen, csak a hiba voltát fedezzük fel
 - fehér doboz tesztelés*: a programrész teljes mértékben ismert, tehát így a hiba helyét is megtalálhatjuk
 - a tesztelés módszere lehet *statikus* (kód kiértékelés és ellenőrzés), illetve *dinamikus* (futtatás adott tesztesetekkel)
- ELTE IK, Alkalmazott modul: Programozás 1:29

- Szoftverfejlesztés**
A szoftverfejlesztés optimalizálása
- A szoftverfejlesztés során a legfőbb cél, hogy a kész program megfeleljen a *funkcionális és minőségi követelményeknek*
 - emellett, a fejlesztők számára fontos, hogy a kész szoftver fejlesztése a lehető legoptimálisabb legyen
 - A szoftverek tervezésének és programozásának módszerét nevezzük *programozási paradigmának*
 - meghatározza a programozási stílust, az absztrakciós szintet
 - meghatározza az alkalmazható programozási nyelvek körét is, és fordítva
 - sok programozási nyelv több paradigmát is támogatnak, ezek a *multiparadigma* nyelvek
- ELTE IK, Alkalmazott modul: Programozás 1:30

Programozási paradigmák

A paradigmák csoportosítása

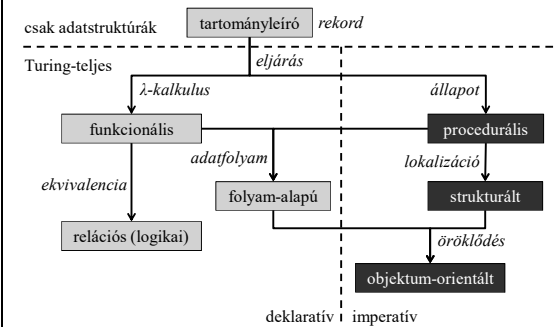
- A programozási paradigmákat két csoportba soroljuk:
 - imperatív:** a programot állapotváltozások sorozatával írja le
 - az utasításokat szekvenciálisan hajtja végre
 - megfelel a gépi szintű végrehajtásnak
 - deklaratív:** a program a tartalmát, megjelenését írja le, nem pedig a funkció végrehajtásának módját
 - nem alkalmaz változókat, csak konstans értékeket
 - az utasításokat nem feltétlenül szekvenciálisan hajtja végre, automatikusan párhuzamosít
 - magasabb szintű kifejezőerővel rendelkezik

ELTE IK, Alkalmazott modul: Programozás

1:31

Programozási paradigmák

A jelentősebb paradigmák



ELTE IK, Alkalmazott modul: Programozás

1:32

Programozási paradigmák

Procedurális programozás

- Procedurális (Procedural):**
 - a programot *alprogramokra* (*subroutine*) bontja, és minden alprogram meghatározott részfeladatot végez el
 - az alprogramoknak két típusa lehet:
 - eljárás (procedure):** valamilyen utasítássorozatot futtat, végeredmény nélkül
 - függvény (function):** valamilyen matematikai számítást végez el, és megadja annak eredményét
 - az alprogramok programkonstrukciókkal épülnek fel, meghívhatnak más alprogramokat, és kommunikálhatnak velük

ELTE IK, Alkalmazott modul: Programozás

1:33

Programozási paradigmák

Procedurális programozás

- a vezérlést a főprogram szolgáltatja, amely kezeli a teljes programban jelen lévő adatokat
- nyelvek: *Fortran, C, BASIC, Pascal*
- pl. (C, vektor összegzése függvénnyel):


```
int sum(int[] values, int size){
    // a függvény paraméterben megkapja a vektort
    int sum = 0;
    for (int i = 0; i < size; i++){
        sum = sum + values[i];
    }
    // ciklussal hozzávesszük ez elemeket
    return sum; // visszatérési érték az összeg
}
```

ELTE IK, Alkalmazott modul: Programozás

1:34

Programozási paradigmák

Procedurális programozás

- pl. (Fortran, tömb átlaga):


```
implicit none
integer :: number_of_points real,
dimension(:), allocatable :: points
real :: average_points=0.

read (*,*) number_of_points
allocate (points(number_of_points))
read (*,*) points
if (number_of_points > 0)
    average_points = sum(points)/number_of_points
deallocate (points)
write (*,'('Average = ', 1g12.4)')
    average_points
```

ELTE IK, Alkalmazott modul: Programozás

1:35

Programozási paradigmák

Procedurális programozás

- pl. (Maple, szám faktoriálisa):


```
myfac := proc(n::nonnegint)
    local out, i;
    out := 1;
    for i from 2 to n do
        out := out * i
    end do;
    out
end proc;
```

ELTE IK, Alkalmazott modul: Programozás

1:36

Programozási paradigmák

Strukturált programozás

- *Strukturált (Structured)*:
 - a program részegységekre (*csomagokra*, vagy *blokkokra*) tagolódik, minden egység rendelkezik belépési ponttal, és kilépési ponttal
 - a programegységeknek van egy kívülről látható része (*interfész*), és egy belső megvalósítása (*implementáció*)
 - a programban használt adatstruktúrák a programegységeknek megfelelően strukturálódnak
 - támogatja a kivételkezelést, tiltja a programkódban történő ugrást (*goto*)
 - nyelvek: *Pascal, C, ADA*

ELTE IK, Alkalmazott modul: Programozás

1:37

Programozási paradigmák

Strukturált programozás

- pl. (Ada, verem csomag):

```
package STACK_T is
  type Stack is limited private;
  procedure Push (v: in out Stack; e: Value);
  procedure Pop (v: in out Stack);
  function Top (v: Stack) return Value;
private
  type Node;
  type Pointer is access Node;
  type Node is record
    data: Value; next: Pointer := null;
  end record;
  ...
end STACK_T;
```

ELTE IK, Alkalmazott modul: Programozás

1:38

Programozási paradigmák

Strukturált programozás

- pl. (C++, verem típus):

```
class Stack {
private: // rejtett rész, implementáció
  int* values; // attribútumok
  int top;
public: // látható rész, interfész
  Stack(){ values = new int[10]; top = 0; }
  // konstruktor
  ~Stack() { delete[] values; } // destruktor
  void Push(int v) { // metódus
    if (top < 10) { values[top] = v; top++; }
  }
  ...
};
```

ELTE IK, Alkalmazott modul: Programozás

1:39

Programozási paradigmák

Objektum-orientált programozás

- *Objektum-orientált (Object-oriented)*:
 - a feladat megoldásában az alulról-felfelé megközelítést alkalmazza, alapja az *egységbe záras* és az *öröklődés*
 - a programot egymással kommunikáló objektumok alkotják, amelyek valamilyen relációban állnak egymással
 - manapság a legnépszerűbb programozási paradigma, a programozási nyelvek jelentős része támogatja
 - objektumorientált támogatással rendelkező nyelvek: *C++, Objective-C, Matlab, PHP, Python, Perl, ...*
 - tisztán objektumorientált nyelvek: *Smalltalk, Java, C#, Eiffel, Ruby, ...*

ELTE IK, Alkalmazott modul: Programozás

1:40

Programozási paradigmák

Objektum-orientált programozás

- pl. (C++, grafikus felületű ablak):

```
class DemoWindow : public QWidget {
  // ablak osztály
public:
  DemoWindow(QWidget* parent = 0) {
    // a konstruktor megkaphatja a szülőt
    setBaseSize(200, 120);
    setWindowTitle("Demo Window");
    qButton = new QPushButton("Quit", this);
    // gomb példányosítása
  }
private:
  QPushButton* qButton; // gomb az ablakon
};
```

ELTE IK, Alkalmazott modul: Programozás

1:41

Programozási paradigmák

Objektum-orientált programozás

- pl. (C#, weblap vezérlője):

```
public class HomeController : Controller {
  // alapéretelmezett vezérlő
  private MyEntities entities;
  // adatbázis kapcsolat

  public ActionResult List(){
    // akcióművelet
    return View(entities.People.ToArray());
    // adatok lekérése az adatbázisból
  }
  ...
}
```

ELTE IK, Alkalmazott modul: Programozás

1:42