

ADATSZERKEZETEK – TÍPUSOK

DATA STRUCTURES – DATA TYPES

Pap Gáborné, Papne@ludens.elte.hu

Szlávi Péter, Szlavi@ludens.elte.hu

ELTE TTK Informatika Szakmódszertani Csoport

Abstract

In this lecture we are making an attempt to outline the curriculum as well as the educational theory lying behind that we apply in designing the second semester of the course called *Programming Methodology* for students of Information Science at Eötvös Loránd University. (The given semester concentrates on data.) A main characteristic of programming in algorithmic languages is that data have a specific, inevitable attribute called *type*.

Type is a keyword notion in programming i.e. Neumann-based programming. Therefore, it is vital for all programmers to learn how to create new types that can be safely reused in the future. In order to precisely define types we will use an algebraic and a more traditional algorithmic language.

Összefoglaló

Előadásunkban megkísérlünk képet festeni arról az anyagról, oktatási elképzelésről, amelyet egyetemünk Informatika szakjának Programozásmódszertan című tárgyában, annak adatokat középpontjába helyező félévében oktatunk. Az algoritmikus nyelveken történő programozás egyik legfontosabb jellemzője, hogy az adatoknak különös „ismertetőjele”, nélkülözhetetlen velejárója, állapothatározója az ún. *típus*.

A programozás –amely alatt most az ún. Neumann-elvű programozást értjük– számára a típus fogalma kulcs fogalom. Ezért annak megismerése, hogy miként hozhatunk létre biztonságosan és –lehetőség szerint– később is újra felhasználhatóan újabb és újabb típusokat, fontos minden programozással foglalkozó szakember számára. A típus precíz megragadásához felhasználunk egy *algebrai* és egy hagyományosabbnak mondható *algoritmikus nyelvezetet*.

ADATSZERKEZETEK – TÍPUSOK

Pap Gáborné, Papne@ludens.elte.hu

Szlávi Péter, Szlavi@ludens.elte.hu

ELTE TTK Informatika Szakmódszertani Csoport

0. Bevezető

Előadásunkban megkísérlünk képet festeni arról az anyagról, oktatási elképzelésről, amelyet egyetemünk Informatika szakjának Programozásmódszertan című tárgyában, annak adatokat középpontjába helyező félévében, a másodikban oktatunk. Az algoritmikus nyelveken történő programozás¹ egyik legfontosabb jellemzője, hogy az adatoknak különös „ismeretjel”, nélkülözhetetlen velejárója, állapotthatározója az ún. **típus**. [SzZs1]

A típus fogalmával nem ebben a félévben találkozunk először a hallgatók. Van róla elképzelésük, intuitív fogalmaik. [Sz1] Ezek azonban „rendezetlenek”, néha elmosódottak. Sokszor nem tisztán válik szét fejükben az adat és a típus fogalma; programjaikban keverednek; szerencsés esetben szintaktikai hibák formájában jelentkeznek. Erre magyarázatot adhat a programozási rutintalanság, de –érzésünk szerint– az **absztrakciós készségük** még nem megfelelő volta is.

Az előző félévben megismertkedtek az **algoritmikus absztrakciónak** több szintjével. (Strukturált programozás elemei, programozási tételek [SzZs2] és programtranszformációkon alapuló formális építkezés [Sz2, HSZs].) Képesek feladatokat több-kevesebb sikerrel formálisan is specifikálni. Tehát elindultak a formális algoritmikus gondolkodás útján. E, második félévben a programozás másik fontos pillérét vesszük elő: az adat-típusokat. A megkezdett „formális hozzáállást” folytatjuk, a típusok definiálásának igényével. Azt várjuk ettől, hogy egyrészt absztrakciós készségük erősödik majd, másrészt a **moduláris programozásban**, sőt az **objektum-elvű programozásban** is kellő tapasztalatra tesznek ez által szert.²

A beszámolóink gerincét azon egyetemi előadásunk alkotja, amelyet az informatika tanárszakosok számára a típus formális leírásának módszereiről, nyelvezetéről szoktunk tartani.

1. Mi a típus?

Az adatoknak az a tulajdonsága, hogy **értékei** mely **halmazból** származnak (**értékhalmaz**) és **tevékenységeknek** (eljárások függvények, operátorok) mely **„készlete**, amely **létrehozza, felépíti, lerombolja, és részekre bontja**”, alkalmazható rá (**asszociált műveletek**).

¹ Mivel előadásunkban csak az ún. **algoritmikus programozást** érintjük, ezért a továbbiakban röviden csak programozást mondunk, ami alatt algoritmikus programozást értünk.

² Nem kifejezett célja a tárgynak az OOP paradigma elsajátíttatása, de mivel oktatás során nincs kibékíthetetlen ellentét az adattípusokra épülő programfejlesztés és az OOP között, ezért szinten magától érlelődik meg a hallgatóban jónéhány OOP-ben megfogalmazott módszertani elv.

Ahhoz, hogy teljesnek mondhassuk ismereteinket a vizsgált típusról, továbbiakra is szükségünk van. A műveletek használatának mikéntjéhez és várható hatásának figyelembevételéhez még két dologra van szükség:

- **Szignatúra** – milyen dolgokkal manipulál a művelet (*értelmezési tartomány*), s mit állít elő (*értékkészlet*)? Egyszerűen szólva: ez a művelet **szintaktikájának definíciója**, amely megadja a „megszólításának” helyes módját.
- A **műveletek „(együtt)működése”** – hogyan függnek össze a műveletek transzformációi (*axiómák*)?

A művelet működésének, azaz **szemantikájának** definiálására többféle mód van. Ezek más-más absztrakciós szinten ragadják meg ugyanazt, ezért mi mindegyiket felhasználjuk a típus biztonságos megközelítésére, természetesen egymásra épülően. [KV]

- **Csak a kapcsolatukat** definiáljuk, egymásra hatásukat, axiómák megadásával. (**Algebrai specifikáció**)
- Definiáljuk az egyes műveletek „**egyedi**” **leképezését**. Ez az *export modul* feladata, amely tartalmazza az algoritmikus *szignatúrákat*³ és –esetleg– az *elő-utófeltételeket*. (**Algoritmikus specifikáció**)
- Rögzítjük az egyes műveletek **megvalósítását**, amelynél már figyelembe vesszük a típus addigra már rögzített ábrázolását. Ez a **reprezentációs-implementációs modul** feladata.⁴

A fentieket egyaránt vonatkoztathatjuk **algoritmusra** [Sz3] és konkrét **programozási nyelvre** [MM, SW, W]. Egy későbbi fejezetben az algoritmikus nyelvünket úgy bővítjük ki, hogy e kérdéskör kezelhető legyen benne. Mivel előadásunkban nincs módunk a kódolás, témánk számára fontos részleteiben elmélyedni, megállunk a pusztán „elmélet” síkján.

2. A típus algebrai specifikációjáról

Röviden utalunk a nyelv azon kiegészítésére, amelyben tisztázzuk az adott típussal szembeni „globális” elvárásainkat. Ezt a legabsztraktabb elképzelést formálisan egy algebrai alapú nyelven írjuk le, az alábbi keretben:

Szintaktikai „minta”	Magyarázat
Típus <i>TípusKonstrukció</i> (paraméterek) : Asszociált műveletek: <i>Operáció(értelmezési tartomány): Értékkészlet</i> ...	A definiálandó típuskonstrukció feje. Operációinak szignatúrája (értelmezési tartománya, értékkészlete)

³ Tulajdonképpen az eljárás/függvény/operátor algoritmikus nyelven megadott **fejsorát** jelenti.

⁴ Éppúgy teszünk, mint korábban, azaz: a feladatmegoldást a feladat/program specifikálásával kezdtük, hogy ezen ismereteinkre építve készítsük el a program működésének tervét az algoritmikus nyelvünkön. [Sz2, HSzZs] Hasznos analógia:

feladat/program-specifikáció	⇔	típus-specifikáció
↓		↓
algoritmikus program	⇔	típusmegvalósítás = modul.

Axiómák:

Jelölések:

- *objektum: Típus (paraméterek) =*
értelmező megjegyzés
- ...

1° az axióma megnevezése (esetleg informális szövege)
a formalizált axióma

2°

Az operációk kapcsolatát leíró axiómák, és a bennük fölhasznált objektumok „deklarációi”.

Esetleg kibővíthető fontos bizonyítható, bizonyítandó következményállítással.

Megjegyzés:

- ✱ A típus korábban emlegetett értékalmazát a 0-változós konstans operációk alkotják, valamint a belőlük a konstrukciós műveletekkel transzformálható további elemek alkotják. Tehát az értékalmazat *konstruktív*, az előbbi értelemben.

Az algebrai leírás bemutatására álljon itt példaként a hét napjainak típusa:

Típus TNap:

Asszociált műveletek:

Hétfő, Kedd, Szerda, Csütörtök, Péntek, Szombat, Vasárnap,

Min, Max: TNap

Számosság: Egész⁵

Létrehoz: TNap

Következő (TNap) : $TNap \cup \{NemDef\}$

$$\text{Előző}(\text{TNap}) : \text{TNap} \cup \{\text{NemDef}\}$$

Sorszám (TNap) : Egész

$$\text{TNap (Egész)} : \text{TNap} \cup \{\text{NemDef}\}$$

< (T Nap, T Nap) : Logikai

```
= (TNap, TNap) : Logikai
```

Axiómák:

- $a : TNap$
- $n : Egész$

0° Konstans-axióma

Számosság=7 $\wedge$ Min=Hétfő $\wedge$ Max=Vasárnap

1° Létrehozás-axióma

$a = \text{Létrehoz} \Rightarrow a = \text{Hétfő}$

2° Előző-Következő axióma

$$\exists \text{Előző}(a) \Rightarrow \text{Következő}(\text{Előző}(a)) = a \wedge$$
$$\exists \text{Következő}(a) \Rightarrow \text{Előző}(\text{Következő}(a)) = a$$

3° Előző-Következő hiba-axióma

$$El\ddot{o}z\ddot{o}(H\acute{e}tf\ddot{o})=NemDef \wedge K\ddot{o}vetkez\ddot{o}(Vas\acute{a}rnap)=NemDef$$

⁵ A konstansokat (Hétfő, ..., Min, Max, Számosság) 0-változós függvényeknek tekintjük. Közülük a Min/Max-t minden rendezett típushoz is, a Számosságot minden végeshez is hozzá szoktuk rendelni. Így polimorfiát hoztunk létre ezen operációkra nézve, amely majd egyéni intézkedést kíván a következő specifikációs lépésnél.

4° Sorszám-axióma
 $Sorszám(Hétfő) = 0 \wedge Sorszám(Vasárnap) = 6$

5° Sorszám-TNap axióma
 $n \in [0..Számosság) \Rightarrow Sorszám(TNap(n)) = n \wedge TNap(Sorszám(a)) = a$

6° TNap hiba-axióma
 $n \notin [0..Számosság) \Rightarrow TNap(n) = NemDef$

7° Következő-axióma
 $Következő(Hétfő) = Kedd \wedge Következő(Kedd) = Szerda \dots^6$

8° Rendezési axióma
 $a < b \Leftrightarrow Sorszám(a) < Sorszám(b) \dots$

9° Azonosság-axióma
 $a = b \Leftrightarrow Sorszám(a) = Sorszám(b) \dots$

10° Következő-Sorszám axióma
 $\exists Következő(a) \Rightarrow Sorszám(Következő(a)) = Sorszám(a) + 1$

1. Állítás:
 $Sorszám(Kedd) = 1 \wedge \dots$

Bizonyítás:
 $\dots$

2. Állítás:
 $(Min < a \vee Min = a) \wedge (a < Max \vee a = Max)$

Bizonyítás:
 $\dots$

Értelmező megjegyzések:

- Megfigyelhetjük, hogy mennyi információt hordoz a műveletek szignatúrája. (Pl. a **Létrehoz** értelmezési tartomány nélküli függvény, amely a „semmitől” kreálja meg az adott típusú adatot.)
- Ritka pillanat tanúi vagyunk: a típus értékhalmozát explicite adjuk meg, elemeinek felsorolásával (nem generálásával).
- A **NemDef** egy típusfüggetlen konstans, ami akkor képződik, amikor valamilyen tiltott művelet végrehajtására kerülne sor. Ha egyszer **NemDef** érték jött ki, akkor arra alkalmazva bármely operációt, az eredmény **NemDef** marad (**implicit error-axióma**).
- A 8° axiómában láthatjuk, hogy miként vezetjük vissza a $<_{TNap}$ rendezést a $<_{Egész}$ rendezésre.⁷ A 9°-ben pedig a $=_{TNap}$ rendezést a $=_{Egész}$ azonosságra.
- Az algebrai leírás „legfőbb” pedagógiai haszna, hogy jól elemezhetővé válik általa a típus: műveleteinek elégségessége, teljessége, ellentmondásmentessége, s esetleges minimalitása.

3. A modul

Most hogy a típus absztrakt tudnivalóit összeállítottuk és lejegyeztük –mondhatjuk: specifikáltuk a típust–, jöhet a programozás klasszikus lépései közül az algoritmikus tervezés.

⁶ Ennek párjaként megfogalmazható „Előző-rendezési axiómát” következmény állításként illeszthetjük ide.
 (Most nem tesszük meg.)

⁷ $<_H: H \times H \rightarrow \text{Logikai}$, és teljesíti a rendezéssel szemben támasztott szokásos feltételeket.

Annak érdekében, hogy minél inkább időt állót alkossunk –ahogy mondani szokták: újra-felhasználható– két lépésben végezzük ezt el. Elsőként azt a „felületet” rögzítjük, amelyen keresztül a típus majdani felhasználója illeszkedni képes a típushoz, azaz amilyen szintaxisal hivatkozhat a típus legkülönbözőbb fogalmaira (export modul). Ezután foglalkozunk a megvalósítás tényleges részleteivel (reprezentációs-implementációs modul). Ez utóbbit akár többféle változatban is elkészíthetjük.

Elsőként a programozói felületet leíró, ún. **export modul** szintaxisát adjuk meg:

```
ExportModul TípusNév (InputParaméterek) :
    [most következnek az exportálandó fogalmak „értelmesen” csoportosított felsorolása:]
Típus    [ritkán van ilyenre szükség]
    Tip1, Tip2, ... [csak azonosítók felsorolása!!!]
Konstans [ritkán van ilyenre szükség]
    Kons1, Kons2, ... [a TípusNév típus néhány kiemelendő konstansa]
    [a típushoz tartozó tevékenységek „fejsorai” és működésüket leíró elő-utófeltételek:]
Függvény Fv1 (FormParam) : FvTip1
    Ef: ... előfeltétel ...
    Uf: ... utófeltétel ...
    ...
Eljárás Elj1 (FormParam)
    Ef: ...
    Uf: ...
    ...
    ...
Modul vége.
```

Megjegyzések:

- A „fő fogalom”, a **TípusNév**, külön megadás híján is látszódo fogalom lesz.
- Az **InputParaméterek** lehetnek típust jellemző konstansok, de –típuskonstrukció esetén– típusok is. (Szokás e paramétereket ’generic parameter’-nek nevezni.)
- A **modulfej** egyben a felhasználás „mintájául” is szolgál.

Kiegészítjük az algebrai specifikáció műveletkészletét továbbiakkal is: **egyenlőség-vizsgálattal**, **értékadással**, **beolvasás** és **kiírás** műveletekkel, valamint a hibás műveletvégzés tényét rögzítő **hibaflag-et kezelő függvénnyel**. (Ezek nélkül ugyanis felhasználhatatlan eszköz maradna a legtöbb típus, vagy minden minden egyes esetben a programozónak külön kellene jól-rosszul e műveletekkel kiegészítenie.)

Egy-két típusnál szokatlan lehet valamely **létrehozó**, ill. **leromboló** művelet használatának explicit felkínálása, mivel a programozási nyelvek legtöbb implementációja *automatikusan* gondoskodik a létrehozásról. (Pl. Tömb.) A leírás teljességére való törekvés, illetve az esetleges „hagyományostól” eltérő ábrázolás lehetővé tétele mégis indokoltá tehetik ezt.

Nézzük visszatérő példánk egy lehetséges export modulját!

```
ExportModul TNap: [kiindulási alap az algebrai leírás]
Konstans
    Hétfő, Kedd, Szerda, Csütörtök, Péntek, Szombat, Vasárnap: TNap
```

```

Operátor Min:TNap
  Másnéven Min' TNap
  Ef: -
  Uf: Min=Hétfő

Operátor Max:TNap
  Másnéven Max' TNap
  Ef: -
  Uf: Max=Vasárnap

Operátor Számosság:Egész
  Másnéven Számosság' TNap
  Ef: -
  Uf: Számosság=7

Függvény Következő(Változó x:TNap):TNap
  Ef: x≠Vasárnap
  Uf: Következő(Hétfő)=Kedd ∧ ...

Függvény Előző(Változó x:TNap):TNap
  Ef: x≠Hétfő
  Uf: Előző(Kedd)=Hétfő ∧ ...

Függvény Sorszám(Konstans x:TNap):Egész
  Ef: -
  Uf: Sorszám(Hétfő)=0 ∧ ...

Függvény TNap(Konstans x:Egész):TNap
  Ef: x∈[0..6]
  Uf: TNap(0)=Hétfő ∧ ...

...

Operátor Be(Változó x:TNap)
  Másnéven Be:x
  Ef: Input∈{'hétfő','kedd','szerda'...}
  Uf: Input='hétfő' ⇒ x=Hétfő ∧

Operátor Ki(Konstans x:TNap)
  Másnéven Ki:x
  Ef: -
  Uf: x=Hétfő ⇒ Output='hétfő' ∧

Infix Operátor Kisebb(Konstans x,y:TNap):Logikai
  Másnéven x<y
  Ef: -
  Uf: Sorszám(x)<Sorszám(y)

Infix Operátor Azonos(Konstans x,y:TNap):Logikai
  Másnéven x=y
  Ef: -
  Uf: Sorszám(x)=Sorszám(y)

Függvény Hibás?(Változó x:TNap):Logikai
  [teszteli, hogy volt-e hibásan elvégzett művelet (pl. amelynek az előfeltétele
  nem teljesült) az adott TNap típusú adatra; majd inicializálódik a hibaflag]

```

Modul vége.

Megjegyzés:

- A **Min**, **Max** és **Számosság** konstansokat *0-változós* (prefix) *operátorként* definiáltuk, s nem konstansként, mert alkalmazásuk speciális szintaxisú.
- Akkurátusan törekedtünk arra, hogy az *utófeltételekben ne nyúljunk vissza más művelethez* (a ∞ -rekurzió elkerülésére), csak saját „hatáskörben” definiáltuk az elvárásokat. Természetesen a „...” folytatás jelzése formálisan *megengedhetetlen*, azonban itt példáról van szó, így megelégedtünk a kitalálható folytatás jelzésével. Mivel így tettünk, fölvethető a kérdés: mennyire *ugyanarról szól* a kétféle specifikáció. Valóban ez külön vizsgálandó, sőt bizonyítandó probléma!
- Az *előfeltétel* gyakorta triviálisan származtatható az axiómákból (l. a megfelelő művelet hiba-axiómája).
- A „**Be:**” és „**Ki:**” operátorok működésénél két problémát kell tudni formalizálni:
 1. a *külvilággal való kapcsolatét* (amely karakteres alapú),
 2. *Szöveg* $\leftrightarrow$ *TNap* *transzformációét*.

Az 1. általános megoldására bevezettünk két függvényt. Az **Input** megadja azt a szöveget, amelyet a beolvasó utasítás talál az input-pufferben; az **Output** szimbolizálja az output-puffer állapotát (azt a *szöveget*, amely kiíródik a végrehajtás során, pl. a képernyőre). Így a 2. probléma már kezelhetővé vált, hisz az adat és transzformáltja is egy-egy memóriabeli objektumban csücsül. Fel kell figyelni a szöveges és „belső-állapotú” konstansok merev megkülönböztetésére ('*hétfő*' $\neq$ *Hétfő*...)!

- **Operátor**nak azt az alprogramot nevezzük, amely az eljárástól vagy függvénytől *eltérő szintaxissal* építhető be a programba (azaz hívható). Lehet **Infix**, ekkor a két operandus *közé* illeszkedik, lehet **Prefix**, ekkor *megelőzi* az egyetlen paraméterét, ill. **Postfix** esetben *követi*. S lehet „egyéni” szintaxisa, ekkor a **Másnéven** kulcs-szónál adhatjuk meg ezt. A leggyakoribb eset a prefix, ezért ezt a minősítőt elhagyhatjuk.

Folytassuk a megvalósítást betetőző fogalommal, a *reprezentációs-implementációs modul*-al, röviden a modullal! Ennek feladata, hogy amit eddig elterveztünk, azt kidolgozza: azaz tisztázza, hogy

- miként *ábrázolható* a típusbeli érték a memóriában, vagy bárhol (*reprezentáció*) és
- hogyan *valósíthatók meg* az egyes műveletek, figyelembe véve a választott ábrázolást (*implementáció*).

A modul-szintaxis a következő:

Modul *TípusNév* (InputParaméterek) :

Reprezentáció

...

[a típusábrázoláshoz szükséges adatleíró „kellékek”: konstansok, saját típusok, a változó-deklarációs rész már magának a típusnak a „mezőit” határozza meg]

Implementáció

[itt szerepelnek az export-modulban „megjósolt” tevékenységek eljárásai...]

```

Függvény Fv1(FormParam) : FvTip1 8
  Ef: ...
  Uf: ...
  ...
Inicializálás
  [egy ilyen típusú adat létrehozása az itt leírt utasításokkal történik]
Modul vége.

```

Megjegyzés:

- Az egyes műveletek egyedi specifikációját jelentő elő-utófeltétel párok, amennyiben megegyeznek az exportmodulbelivel, nyilván elhagyhatók. Természetesen „értelmesen” igazítva a reprezentációhoz, újabb biztonsági támpontul szolgálhat a fejlesztéshez.

És a [visszatérő példánk](#) egy lehetséges moduljának töredéke:

```

Modul TNap: [kiindulási alap az export modul]
Reprezentáció
  Változó
    felsKód:Egész
    hiba:Logikai
  Konstans
    Hétfő:(0,Hamis)      [1. 4° axiómát!]
    Kedd:(1,Hamis)      [1. 1. állítást!]
    ...
    Vasárnap:(6,Hamis) [1. 4° axiómát!]
Implementáció
  Operátor Min:TNap
    Másnéven Min' TNap
    Min.felsKód:=Hétfő.felskód
  Operátor vége.
  ...
  Függvény Következő(Változó x:TNap):TNap
    Ef: felsKód≠Vasárnap.felskód
    Uf: felsKód=Hétfő.felskód ⇒ Következő(x).felsKód=Kedd.felskód
        ^ ...
        [egyszerűbben is lehet a 10° axióma miatt:
        Következő(x).felsKód=x.felskód+1]
    Ha felsKód=Vasárnap.felskód akkor
      hiba:=Igaz
    különben
      Következő.felsKód:=felsKód+1 [1. 10° axiómát!]
    Elágazás vége
  Függvény vége.
  ...
  Függvény Hibás?(Változó x:TNap):Logikai
    Hibás?:=hiba; hiba:=Hamis
  Függvény vége.

```

⁸ Ehhez hasonlóan itt kerülnek deklarálásra a típus eljárásai, illetve operátorai (azaz a sajátos szintaxisú eljárásai és függvényei).

```
...
Inicializálás
    hiba:=Hamis; felsKód:=Hétfő.felskód
Modul vége.
```

Megjegyzés:

- Egyik-másik operátornál meglepő a paraméter **hozzáférési joga**. A várható konstans helyett változó lett azért, mert a végrehajtás során hiba következhet be, amely által az „állapota” kényszerűen és szándékunk ellenére megváltozik.
- Figyeljünk föl rá, hogy az **utófeltételben** sok esetben **nem rögzítjük a hiba-flag értékét!** Ez szándékos: nincs változás!
- Időnként attól a **konvencióntól**, hogy a típus reprezentációjában szereplő komponensekre (felsKód, hiba) nem a rekordnál szokásos mezőhivatkozással jelöléssel tesszük meg (x.felsKód helyett csupán felsKód), **eltérünk**, az a magyarázata, hogy az adott operációnál nem lenne egyértelmű, hogy melyik ilyen típusú adat valamely komponenséről van éppen szó. Pl. a **Következő** függvényben „szóba kerül” két TNap típusú objektum is: a kiindulásul szolgáló x-paraméter és az eredményt jelentő.
- Vegyük észre: az **inicializálásban** éppen az **1^o axióma** szerint jártunk el, a **Következő** függvényénél a **10^o axióma** szerint; a többiekénél is a kapcsolat tetten érhető, csak azoknál nem ennyire „látványosan”.

Nos ez az a pont, amikor a tervezés után a **kódolás** következik. Ezt a lépést a legkülönbözőbb nyelvek a legkülönbözőbb módon támogatják. A tárgyunkban eddigi gyakorlat szerint a Pascal nyelv szerepelt, amely –bevallhatjuk– sokszor kemény dióként ellenállt. Ok leginkább a modul számára kínákozó **unit** fogalom paraméterezhetetlen volta. Némileg kikerülhető ez a hiányosság az **include** lehetőség felhasználásával, vagy az **objektum-osztály** fogalmának bevezetésével.

4. Záró gondolatok

A fent vázolt gondolatok csak az elvi alapját jelentik annak az építménynek, amelybe beletartoznak olyan fontos szerkezetek, mint a **listák**, **sorok**, **prioritási sorok** és **vermek** [PSzZs], valamint a **táblázatok** és a legkülönbözőbb **file-ok** (s azok általánosításai) [SzZs3]. Erre építjük a későbbi félévekben a **rekurzív típusokat** és a **gráfokat** is. [SzZs4]

Reméljük, hogy sikerült valamelyest felvázolni a típus fogalomra épített programozás-módszertani elképzeléseinket, amelynek célja kettős volt: konkrét támaszt adni a típus- (vagy adatszerkezet-) alapú, moduláris programfejlesztéshez, másrészt az absztrakciós készség fejlesztése.

5. Irodalomjegyzék

[SzZs1] Szlávi Péter, Zsakó László: *Programozási bevezető*. ELTE TTK Informatikai Tan-
székcsoport, 1995

[Sz1] Szlávi Péter : [Programozási tételek szerepe a gyakorlati programozásban](http://info.elte.hu/szlavi/ProgModsz/SpAIKo.pdf). <http://info.
inf.elte.hu/szlavi/ProgModsz/SpAIKo.pdf>, 1996

- [SzZs2] Szlávi Péter, Zsakó László: *Programozási tételek*. ELTE TTK Informatikai Tanszékcsoport, 1996
- [Sz2] Szlávi Péter: [Programtranszformációk](http://izzo.inf.elte.hu/szlavi/ProgModsz/Progran.pdf). <<http://izzo.inf.elte.hu/szlavi/ProgModsz/Progran.pdf>>, 2001
- [HSzZs] Harangozó É.-Szlávi P.-Zsakó L.: *Joining Programming Theorems, a Practical Approach to Program Building*. Annales Universitatis Scientiarum Budapestinensis. Sectio Computatorica 17, 155-172, 1998
- [KV] Kozma László, Varga László: *Adattípusok osztálya*. ELTE TTK Informatikai Tanszékcsoport, 2001
- [Sz3] Szlávi Péter: [Programok, programspecifikációk](http://izzo.inf.elte.hu/szlavi/ProgModsz/progspec.doc). <<http://izzo.inf.elte.hu/szlavi/ProgModsz/progspec.doc>> Informatika a Felsőoktatásban '99, Debrecen, 1999. augusztus 27-29.
- [MM] A. McMonnies, S. McSporran: *Developing Object-Oriented Data Structures Using C++*. McGraw-Hill, 1995
- [SW] D. Stubs, N. Webre: *Data Structures with Abstract Data Types and Pascal*. Brooks/Cole Publishing Co., 1985
- [W] Wayne Amsbury: *Data Structures*. Wadsworth Publishing Co., 1985
- [PSzZs] Pap Gáborné, Szlávi Péter, Zsakó László: *Adattípusok*. ELTE TTK Informatikai Tanszékcsoport, 1998
- [SzZs3] Szlávi Péter, Zsakó László: *Gráfok, gráfalgoritmusok*. ELTE TTK Informatikai Tanszékcsoport, 2000
- [SzZs4] Szlávi Péter, Zsakó László: *Adatfeldolgozás*. ELTE TTK Informatikai Tanszékcsoport, 1996