A módszeres programozás absztrakciós szintjei, a programozási paradigmák és a programozási nyelvek támogatása

Menyhárt László Gábor
(ORCID: 0000-0002-1574-4454)

menyhart@inf.elte.hu
ELTE Eötvös Loránd Tudományegyetem
Informatikai Kar

Absztrakt. Egy hétköznapi probléma számítógépes megoldása során több absztrakciós lépésre is sor kerül. A gyakorlott programozók számára ezek a lépések magától értetődőek, míg a kezdő programozók számára gyakran nehézséget okoznak. Ebben a cikkben összeszedem, hogy a módszeres programozás hogyan segíti az absztrakciót, hogyan jelennek meg a programozási paradigmák, mit segítenek és miben nehezítenek a programozási nyelvekben megjelenő paradigmákat támogató szintaktikai lehetőségek. Egy konkrét feladat megoldásával illusztrálom is a lépéseket és felmerülő problémákat.

Kulcsszavak: módszeres programozás, programozási paradigma, programozási nyelvek, oktatás, mérés

1. Bevezetés

Egy hétköznapi probléma megoldásához nem feltétlenül van szükség számítógépre. Amikor azonban sokszor kell elvégezni valamit, nagy mennyiségen kell ugyanazt megtenni, egyre gyakrabban halljuk a robotizáció bevezetését. Ilyenkor a gépeket programozni kell és a probléma megoldásához szükséges az adatokat, bemeneti paramétereket digitalizálni. Azaz valamilyen módon mérni kell és számítógéppel feldolgozható formába kell hozni. Ez a lépés majdnem mindig kimarad a programozók képzésének elején, rögtön elvárjuk, hogy feladat szövegének elolvasása után képesek legyenek az adatokat kigyűjteni. Ebben a cikkben egy konkrét példán keresztül igyekszem bemutatni a számítógépes problémamegoldás különböző lépései.

2. A probléma felvetése

A feladat

A feladatunk legyen az, hogy egy születésnapi ünnepségen sokféle üdítőital került az asztalra. Valaki kitalálta, hogy a legcukrosabb italokat szedjük le az asztalról. Gyűjtsük ki a legcukrosabb italokat!

A módszeres programozást [5] követve megsejtjük a felhasználandó algoritmusmintákat, ami a jelen esetben a maximumkiválasztás és a kiválogatás. Ezek absztrakt specifikációját és algoritmusát ismerjük, így könnyebben elkészítjük a konkrét specifikációt és az algoritmusokat, amit lekódolunk, majd felismerjük, hogy hatékonyabban is meg tudjuk oldani a feladatot az algoritmusminták összeépítésével.

Az absztrakciós lépések

Az első absztrakciós lépés annak a kitalálása, hogy az adatokat hogyan reprezentáljuk, illetve az adatok megszerzése. Ilyenkor már a feladatot is átfogalmazzuk erre a szintre. A *miből?*, *mit?* és *mi lesz igaz?* kérdésekre válaszolunk. Néha előfordul, hogy az adatokat transzformálni is kell, hogy könnyebben

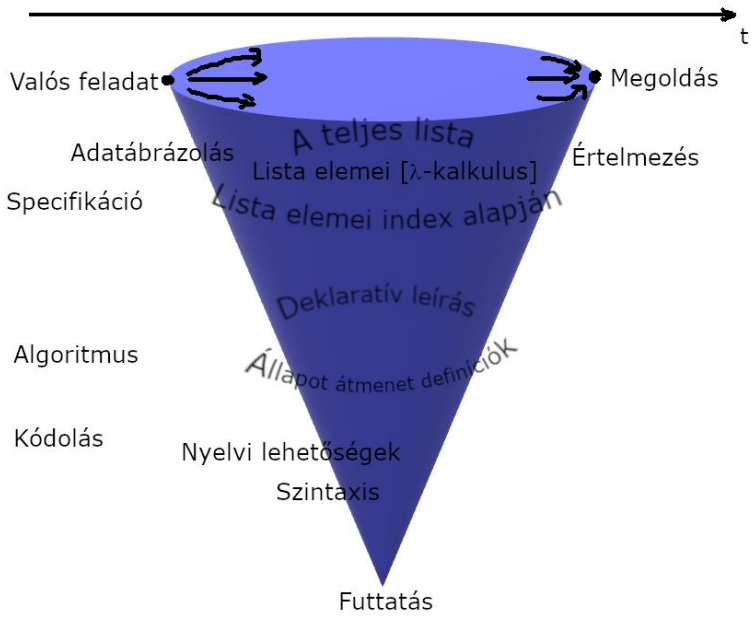
megoldjuk a feladatot. Ezeket az absztrakciós lépéseket segítik a módszeres programozás specifikációs elvárásai, ahol az adatok, típusok és azok megszorításai, azaz igaz matematikai kifejezések felírásával kompakt módon definiáljuk a problémát.

A második absztrakciós lépés sikeres teljesítésével eljutunk arra a szintre, hogy tudunk válaszolni a *hogyan?* kérdésre. A módszeres programozás sémákkal, programozási tételekkel, azaz algoritmusmintákkal segít. Ugyanakkor elképzelhető itt az imperatív állapotátmenetek leírása helyett egy deklaratív leírás is.

A harmadik absztrakciós lépéssel eljutunk az implementációhoz, amit a korábban megfogalmazott algoritmus vagy deklaratív leírás alapján könnyebben készítünk el. Itt arra a kérdésre válaszolunk, hogy *a gép hogyan?* végezze el a feladatát.

Másik oldalról közelítve fontos, hogy hogyan tervezünk, mi a kódolás módszere, milyen a megoldás stílusa, azaz milyen programozási paradigmát követünk. Én most a két fő, imperatív illetve deklaratív paradigmára koncentrálok. Egyre több programozási nyelvben jelenik meg egymás mellett több paradigma, amiket ezért multiparadigma nyelveknek hívhatunk. Konkrétan a λ-kalkulus támogatása jelenik meg, amivel egy nem teljesen tiszta funkcionális programozás hajtható végre, ahhoz hasonló gondolkodást követ.

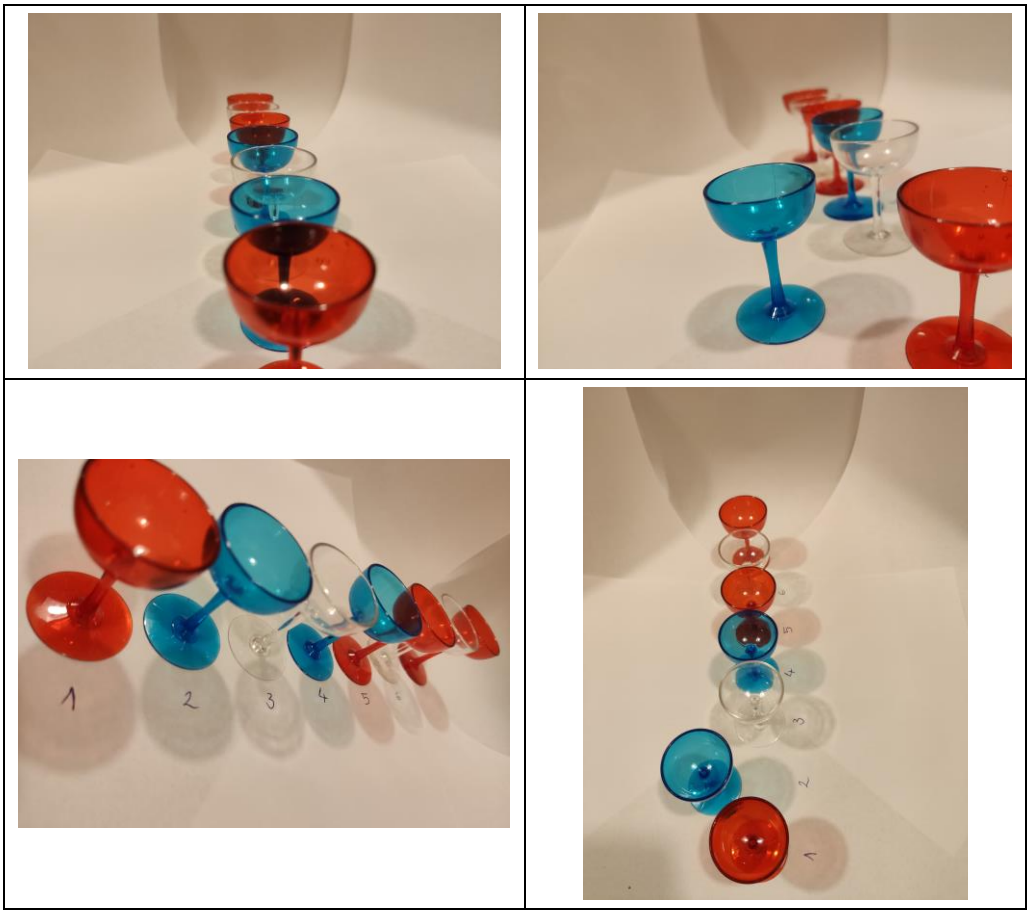
Az érdekel, hogy különböző absztrakciós szinteken hogyan jelenhetnek meg a paradigmák. A szoftvertesztelési V-modellből kiindulva felrajzoltam egy 3D-s alakzatot, mely azt próbálja érzékeltetni, hogy az egyes absztrakciós szinteken hogyan jelennek meg a paradigmák.



1. ábra: Hogyan jutunk el a feladattól a megoldásig?

Egy problémát sokféleképpen elkezdhetünk és a végén – reményeink szerint – ugyanazon megoldáshoz, vagy megoldásokhoz jutunk. Már a valós életben is megjelenhetnek a gondolkozási módok, szempontok. Lehet, hogy a poharak sorozata vagy listája, amire gondolunk, de előfordulhat, hogy az

egyes poharakkal szeretnénk foglalkozni, vagy ismerjük a poharak számát és a sorszámukkal hivatkozunk rájuk. De ez a lényegen, végeredményen nem változtat, csak más nézőpontból figyeljük.



2. ábra: Hogy tekintünk a poharakra?

3. Tervezés

Specifikáció

Ebben a konkrét feladatban is többféleképpen lehet az adatokat ábrázolni. Elképzelhető a nevek listája és külön a cukortartalmak listája azzal a kiegészítő információval, hogy azonos elemszámúak a listák és az azonos pozíciókon az összetartozó adatok vannak; vagy a névből és cukortartalomból álló összetartozó rekordok listáját vesszük. Már ettől a választástól is függ, hogy hogyan tudunk továbbhaladni.

Bemenet:

$N \in \mathbb{N}$

$it \in (n\ x\ c)^N$, italok adatai

$n \in \mathbb{S},\ c \in \mathbb{N}$, név illetve cukortartalom

Kimenet:

$db \in \mathbb{N}$

$lc \in \mathbb{S}^{db}$, legcukrosabbak

Előfeltétel:

nincs

Utófeltétel:

$$\exists j(1 \leq j \leq N): maxC = it_j.c \wedge \forall i(1 \leq i \leq N): it_i.c \leq maxC$$

$$db = \sum_{it_i.c = maxC}^{N} 1$$

$$\forall i\ (1 \leq i \leq db)\ \exists j(1 \leq j \leq N): lc_i = it_j.n \wedge it_j.c = maxC$$

$$halmazfelsorolás\ (lc)$$

Játszunk el kicsit a jelölésekkel és nézzük meg, hogy a paradigmák hogyan jelenhetnek meg a specifikáció szintjén.

Írjuk át az első sort kompaktabb módra:

$$maxC = \underset{i\,=\,1}{\overset{N}{MAX}}\ it_i.c$$

Vagy használhatunk halmazt is:

$$maxC = \underset{ital \in \boldsymbol{it}}{MAX}\ ital.c$$

Halmazjelölésekkel is felírhatjuk az első hosszabb kifejezést:

$$\exists ital \in \boldsymbol{it}: maxC = ital.c \wedge \forall masital \in \boldsymbol{it}: masital.c \leq maxC$$

Vagy itt a legkompaktabb lehetőség:

$$maxC = MAX\ \boldsymbol{it}.c$$

Az előzőek mintájára a szumma jelnél használjunk halmazt:

$$db = \sum_{ital \in \boldsymbol{it}\ ital.c = maxC} 1$$

Kihagyhatunk indexet az egyikből:

$$\forall i\ (1 \leq i \leq db)\ \exists ital \in \boldsymbol{it}: lc_i = ital.n \wedge ital.c = maxC$$

Vagy akár mindből:

$$\forall egyiklc \in lc: \exists ital \in \boldsymbol{it}: egyiklc = ital.n \wedge ital.c = maxC$$

Így itt is kaphatunk kompakt formát:

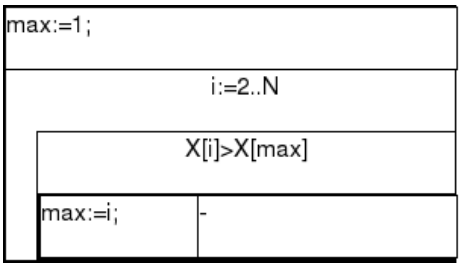
$$\forall egyiklc \in lc: \exists ital \in \boldsymbol{it}: egyiklc = ital.n \wedge ital.c = MAX\ \boldsymbol{it}.c$$

Algoritmus

Használjuk az algoritmusmintákhoz tartozó absztrakt algoritmusokat:

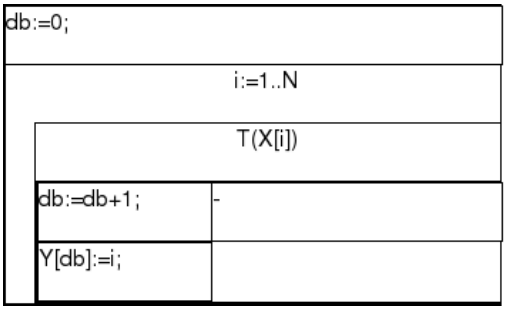
Maximumkiválasztás

```
max:=1
Ciklus i:=2-től N-ig
  Ha X[i]>X[max] akkor
    max:=i
  elágazás vége
Ciklus vége
```



Kiválogatás

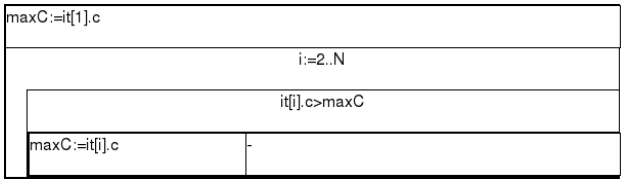
```
db:=0
Ciklus i:=1-től N-ig
  Ha T(X[i]) akkor
    db:=db+1;
    Y[db]:=i;
  elágazás vége
Ciklus vége
```



Írjuk át a konkrét feladathoz tartozó változók és feltételek használatával:

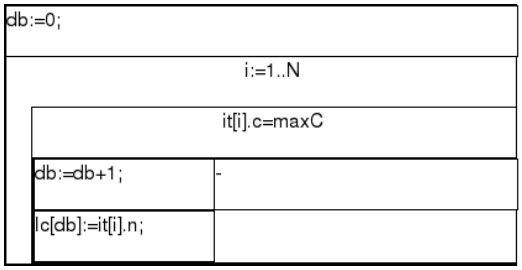
Maximumkiválasztás

```
maxC:=it[1].c
Ciklus i:=2-től N-ig
  Ha it[i].c>maxC akkor
    maxC:=it[i].c
  elágazás vége
Ciklus vége
```



Kiválogatás

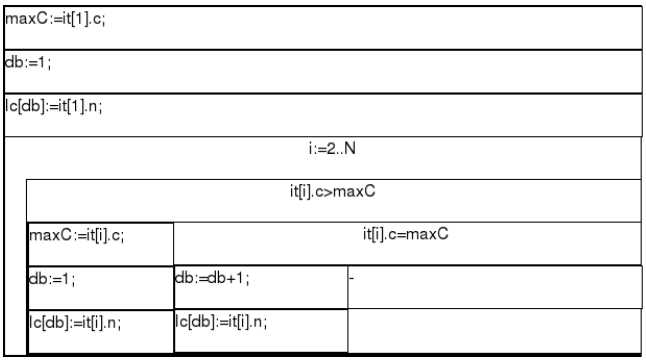
```
db:=0
Ciklus i:=1-től N-ig
  Ha it[i].c=maxC akkor
    db:=db+1;
    lc[db]:=it[i].n;
  elágazás vége
Ciklus vége
```



A tételek összeépítésével hatékonyabb kódot kapunk, így nem kell kétszer megvizsgálni a lista összes elemét:

Maximumkiválasztás és kiválogatás összeépítve

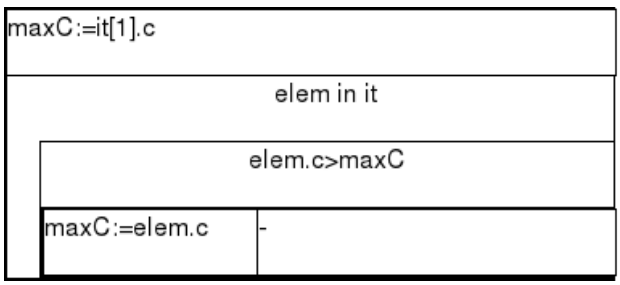
```
maxC:=it[1].c
db:=1;
lc[db]:=it[1].n;
Ciklus i:=2-től N-ig
  Ha it[i].c>maxC akkor
    maxC:=it[i].c
    db:=1;
    lc[db]:=it[i].n;
  különben Ha
it[i].c=maxC akkor
    db:=db+1;
    lc[db]:=it[i].n;
  elágazás vége
Ciklus vége
```



Ugyanezeket átírhatjuk olyan formába, hogy index helyett az egyes elemeket használjuk a ciklus változójaként.

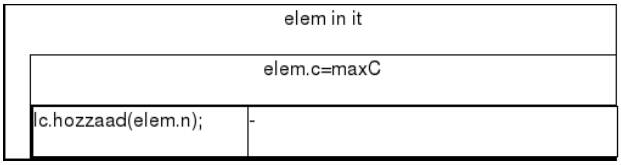
Maximumkiválasztás

```
maxC:=it[1].c
Ciklus elem az it-ben
  Ha elem.c>maxC akkor
    maxC:=elem.c
  elágazás vége
Ciklus vége
```



Kiválogatás

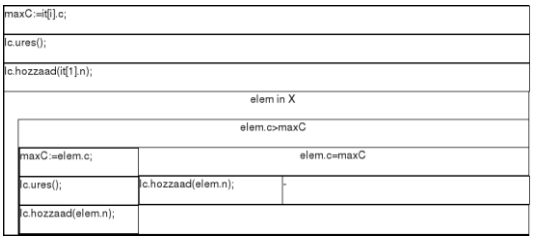
```
lc.üres()
Ciklus elem az it-ben
  Ha elem.c=maxC akkor
    lc.hozzáAd(elem.n);
  elágazás vége
Ciklus vége
```



A tételek összeépítésével itt is hatékonyabb kódot kapunk, amiben nem kell kétszer végighaladunk a lista elemein:

Maximumkiválasztás és kiválogatás összeépítve

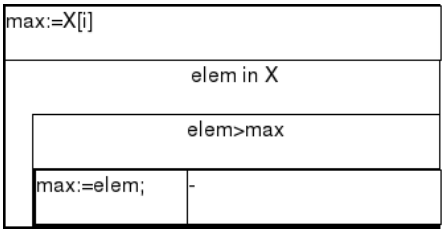
```
maxC:=it[1].c;
lc.üres();
lc.hozzáad(it[1].n);
Ciklus elem az it-ben
  Ha elem.c>maxC akkor
    maxC:=elem.c
    lc.üres();
    lc.hozzáad(elem.n);
  különben Ha elem.c=maxC akkor
    lc.hozzáad(elem.n);
  elágazás vége
Ciklus vége
```



Természetesen az eredeti absztrakt algoritmusokat is átírhatjuk hasonlóképpen:

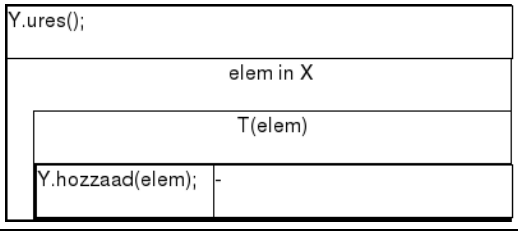
Maximumkiválasztás

```
max:=X[1];
Ciklus elem az X-ben
  Ha elem>max akkor
    max:=elem
  elágazás vége
Ciklus vége
```



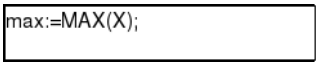
Kiválogatás

```
Y.üres();
Ciklus elem az X-ben
  Ha T(elem) akkor
    Y.hozzáad(elem);
  elágazás vége
Ciklus vége
```



Ez pedig a legkompaktabb algoritmus, amit néhány nyelv támogat is:

```
max:=MAX(X)
```



Ne menjünk el szó nélkül amellett a lehetőség mellett, hogy próbáljuk leírni a sorozatot „egyben". Azaz „látjuk" a sorozat egy-két elemét és „tudjuk", hogy van még a többi elem. Ekkor leírhatjuk, hogy hogyan viselkedik egy rövid sorozat (pl. egy elem) és azt is, hogy hogyan tudjuk visszavezetni rövidebb sorozatra.

```
maximum [e] = e
maximum (e:m:többi) = maximum ( (ha e>m akkor e különben m) : többi )
```

4. Implementáció

Az előző, az ismétlésről szóló cikkem [6] alapján a JavaScriptet [2] választottam az egyik implementációs nyelvnek, mert az támogatja az összes paradigma szerinti kódolási lehetőséget és még kiegészítő függvénykönyvtárak is léteznek hozzá, mint például az Underscore vagy a Lodash. Ezen kívül még Haskell-ben [3, 4], egy tisztán funkcionális nyelven is megírtam a feladathoz tartozó kódokat.

A következő öt kód mindegyike a maximumkiválasztás és kiválogatás egymás utáni alkalmazását implementálja:

```
maxC=it[0].c;
for (let i=1; i<it.length; i++)
{
  if (it[i].c>maxC) {
    maxC=it[i].c;
  }
}
names=[];
for (let i=0; i<it.length; i++)
{
  if (it[i].c==maxC) {
    names.push(it[i].n);
  }
}
```

```
maxC=it[0].c;
it.forEach(elem => {
  if (elem.c>maxC) {
    maxC=elem.c;
  }
});
names=[];
it.forEach(elem => {
  if (elem.c==maxC) {
    names.push(elem.n);
  }
});
```

```
maxC=_.max(_.map(it,function(elem){return elem.c;}));
names=_.filter(it,function(elem){return elem.c==maxC;});
```

```
maxC=_.max(_.map(it,function(elem){return elem.c;}));
names=_.where(it,{c:maxC});
```

```
maxC=lo.max(lo.map(it,function(elem){return elem.c;}));
names=lo.filter(it,function(elem){return elem.c==maxC;});
```

A kódot átírhatjuk úgy, hogy a tételeket összeépítjük így a kód komplexitása nő, megértése nehezebbé válhat, amikor elhagyjuk az algoritmusminta követését, viszont a futása gyorsabb lesz, vagyis nő a hatékonyság. A következő 5 kód ezt mutatja be:

```
maxC=it[0].c;
names=[];
names.push(it[0].n);
for (let i=1; i<it.length; i++)
{
  if (it[i].c>maxC) {
    maxC=it[i].c;
    names=[];
    names.push(it[i].n);
  } else if (it[i].c==maxC) {
    names.push(it[i].n);
  }
}
```

```
maxC=it[0].c;
names=[];
//names.push(it[0].n);
it.forEach(elem => {
  if (elem.c>maxC) {
    maxC=elem.c;
    names=[];
    names.push(elem.n);
  } else if (elem.c==maxC) {
    names.push(elem.n);
  }
});
```

```
maxO=it.reduce((tmpMax,elem) => {
  if (elem.c>tmpMax.maxC) return {maxC:elem.c,names:[elem.n]}
  else if (elem.c==tmpMax.maxC) { newnames=tmpMax.names; new-
names.push(elem.n); return {maxC:tmpMax.maxC,names:newnames};}
  else return tmpMax;
  },
  {maxC:it[0].c,names:[]}
);
```

```
maxO=_.reduce(it,function(tmpMax,elem){
  if (elem.c>tmpMax.maxC) return {maxC:elem.c,names:[elem.n]}
  else if (elem.c==tmpMax.maxC) { newnames=tmpMax.names; new-
names.push(elem.n); return {maxC:tmpMax.maxC,names:newnames};}
  else return tmpMax;
  },{maxC:it[0].c,names:[]}
);
```

```
maxO=lo.reduce(it,function(tmpMax,elem){
  if (elem.c>tmpMax.maxC) return {maxC:elem.c,names:[elem.n]}
  else if (elem.c==tmpMax.maxC) { newnames=tmpMax.names; new-
names.push(elem.n); return {maxC:tmpMax.maxC,names:newnames};}
  else return tmpMax;
  },{maxC:it[0].c,names:[]}
);
```

Haskell-ben csak a két lépéses megoldásra mutatok több példát. Az objektumok listájából kétféleképpen nyerem ki az adatokat:

```
values :: [Integer]
values = [v | (n,v) <- Obj.v ]
```

```
values' = map fieldValue Obj.v
  where
  fieldValue (n,v) = v
```

A maximumkiválasztásra 3 saját implementációt is bemutatok a beépített maximum függvény mellé:

```
maximum' :: Ord a => [a] -> a
maximum' = foldr1 (\x y ->if x >= y then x else y)
```

```
maximum'' :: Ord a => [a] -> a
maximum'' [x]       = x
maximum'' (x:x':xs) = maximum'' ((if x >= x' then x else x'):xs)
```

```
maximum''' :: Ord a => [a] -> a
maximum''' [x]       = x
maximum''' (x:x':xs) | x >= x'   = maximum''' (x:xs)
maximum''' (x:x':xs) | otherwise = maximum''' (x':xs)
```

A nevek kiválogatására is mutatok egy lehetőséget:

```
maxNames m' a = map fieldName (filter (flip equalValue m') a)
  where
  fieldName (n,v) = n
  equalValue (n,v) m = (v==m)
```

Az adatok tárolását bemutató példa és a forráskódok teljes tartalma az A mellékletben található.

5. Mérés

Mérést végeztem a fent bemutatott kódokkal. A futtatáshoz egy Intel® Core™ i7-8750H 2.20GHz-es processzorú, 32GB memóriával rendelkező számítógépet használtam, melyen 64 bites Windows 10 Pro operációs rendszer futott. A JavaScript kódokat a Node v10.16.0 segítségével futtattam, míg a Haskell forráskódok fordításához a The Glorious Glasgow Haskell Compilation System, version 8.10.1 fordítóprogramot használtam.

A következő táblázatokban szereplő konkrét értékek egymáshoz való viszonyát érdemes figyelni.

Először csak egész számokat tartalmazó listából történő maximum érték meghatározásának idejét hasonlítottam össze:

Implementáció	Idő [ms] (100.000)	Idő [ms] (1.000.000)
For	5	6
ForEach	2	10
Reduce	2	10
Math.max(…	1	HIBA
Math.max.apply(	1	HIBA
_.max	1	3
_.reduce	2	4
lodash.max	2	6
lodash.reduce	2	3

A JavaScript-ben lévő beépített Math.max függvény 100.000 elemszámig nagyon gyors, de a fölött már nem működik. A továbbiakban 1.000.000 elemszámmal dolgoztam, hogy jobban látszódjanak az eltérések.

A maximumérték meghatározása az objektumok listájából:

Implementáció	Idő [ms] (1.000.000)
For	7
ForEach	15
Reduce	11
_.max	14 (20*)
_.reduce	7
lodash.max	15 (20*)
lodash.reduce	5

A csillaggal megjelölt implementációkban szükség volt egy map használatára is, aminek a tiszta ideje nagyobb, mint az összeépített futás. Ebből az látszik, hogy a compiler javít a futás teljesítményén és nem mappel át mindent.

Az objektumok listájából a maximumérték meghatározása Haskell kódok segítségével a következő futásidőket adta:

Implementáció	Idő [ms] (1.000.000)
Maximum values	320
Maximum' values	310
Maximum'' values	135
Maximum''' values	20
Maximum values'	390
Maximum' values'	120
Maximum'' values'	300
Maximum''' values'	20

Ez a táblázat bizonyítja, hogy a tisztán funkcionális nyelveknél is függ a futásidő a konkrét deklaratív leírástól. Ez a leírás nagyobb kifejező erővel rendelkezik, viszont nagyobb absztrakciót igényel és kevésbé áll közel a gépi kódhoz.

A következőkben az objektumok listájából már a maximumértékkel rendelkező összes név meghatározása jön JavaScripttel:

Implementáció	Idő [ms] (1.000.000)
For 2x	25
ForEach 2x	22
_.max + _.filter	22
_.max + _.where	95
lodash.max + lodash.filter	27

Látszik, hogy az Underscore `where` függvényénél a kód jól olvasható, de a futásideje háromszorosa a többihez képest.

Egy lépésben hatékonyabb a megoldás:

Implementáció	Idő [ms] (1.000.000)
For	12
ForEach	15
Reduce	14
_.reduce	10
lodash.reduce	9

Végül legyen itt a Haskell implementáció ideje is, ami most csak a második lépést, a kiválogatást tartalmazza. Ebben viszont benne van a kiírás ideje is, ugyanis a Lazy evaluation miatt nem kerülne feldolgozásra az adat.

Implementáció	Idő [ms] (1.000.000)
maxNames	20

6. Didaktikai megfontolások

Az előzőek alapján azt mondhatjuk, hogy módszeres programozás által nyújtott kérdések, sémák, algoritmusminták segítik a kezdő programozók oktatását [1], de szükséges, hogy ne csak az absztrakciós szinteket, hanem több gondolkodási formát, paradigmát megtanítsunk, hiszen a programozási nyelvekben is lehetőség van különböző megvalósítási módokra. Fontos, hogy a rendszert és az összefüggéseket is megértsék a hallgatók. Ne csak egymás hegyén-hátán legyenek az építőkövek, mert fontos a hatékonyság is. Nem csak a kód áttekinthetőségére, érthetőségére, hanem a futási idejére is tekintettel legyünk.

Úgy gondolom, hogy a sémakövetés nem kell, hogy a kreativitás rovására menjen. Meg kell tanítanunk a diákoknak, hogy ne csak kövessék, hanem értsék is, hogy mi történik. Használhatják az elinduláshoz, de tudniuk kell, hogy miért úgy van és keresniük kell a továbblépési lehetőségeket a legjobb

megoldás elkészítéséhez. Később szükségük lehet arra is, hogy ne csak egy adott nyelv támogatását tanulják meg, hanem *mint eszköz* tekintsenek ezekre a lehetőségekre.

Azért, hogy a diákjaim megtanulják a mögöttes tartalmat és ne legyenek lusták gondolkodni, a kezdő programozók oktatásának korai stádiumában még a specifikációban is kérem a hosszabb leírásokat a `MAX`, `halmazfelsorolás` és hasonló rövidítések helyett. Így később nem csak egymás után dobálják a függvényeket, hanem átgondolják a megoldást.

Irodalom

1. Tananyag kezdő programozók számára
   http://progalap.elte.hu/downloads/seged/eTananyag/ (utoljára megtekintve: 2020.11.18.)
2. JavaScript referencia
   https://developer.mozilla.org/en-US/docs/Web/JavaScript (utoljára megtekintve: 2020.11.18.)
3. Haskell hivatalos oldala
   https://www.haskell.org/ (utoljára megtekintve: 2020.11.18.)
4. Haskell függvényosztályok
   https://hackage.haskell.org/package/base-4.14.0.0 (utoljára megtekintve: 2020.11.18.)
5. Zsakó László, Szlávi Péter: Módszeres programozás, Műszaki Könyvkiadó, Budapest (1986)
6. Menyhárt László Gábor: Overview of Repetition, Central-European Journal of New Technologies in Research Education and Practice (2676-9425): 2020. volume 2, number 1, pp 34-75 (2020).
   DOI: https://doi.org/10.36427/CEJNTREP.2.1

Mellékletek

A. JavaScript implementációk

Tesztadatok

data_9.js formátuma:

```
v = [232839,99370,16617,661198,606350,276061,658523,866515,742354];
exports.v=v;
```

obj_14.js formátuma:

```
v = [{n:"n273797",v:273797},
{n:"n216959",v:216959},
{n:"n524988",v:524988},
{n:"n834189",v:834189},
{n:"n569940",v:569940},
{n:"n187511",v:187511},
{n:"n205697",v:205697},
{n:"n14180",v:14180},
{n:"n857387",v:857387},
{n:"n549725",v:549725},
{n:"n620312",v:620312},
{n:"n23331",v:23331},
{n:"n571258",v:571258},
{n:"n855019",v:855019}];
exports.v=v;
```

Maximum érték meghatározása egészek listájából

```
const _ = require('underscore');
const lo = require('lodash');
const data = require('./data_9.js');
let it = data.it;

console.log("Max");

var timer_start;
var maxC;
var timer_end;

console.log("For");
timer_start = Date.now();

maxC=it[0];
for (let i=1; i<it.length; i++)
{
    if (it[i]>maxC) {
            maxC=it[i];
          }
}

timer_end = Date.now();
console.log(maxC);
elapsed=timer_end-timer_start;
console.log("For;time:",elapsed);
```

```
console.log("foreach");
timer_start = Date.now();

maxC=it[0];
it.forEach(elem => {
          if (elem>maxC) {
            maxC=elem;
          }
});

timer_end = Date.now();
console.log(maxC);
elapsed=timer_end-timer_start;
console.log("foreach;time:",elapsed);

console.log("reduce");
timer_start = Date.now();

maxC=it.reduce((tmpMax,elem) => {
            if (elem>tmpMax) return elem
            else return tmpMax;
          },
          it[0]
);

timer_end = Date.now();
console.log(maxC);
elapsed=timer_end-timer_start;
console.log("reduce;time:",elapsed);

/* This part works with max data_100000 / 100.000 element * /
console.log("Math.max(...");
timer_start = Date.now();

maxC=Math.max(...it);

timer_end = Date.now();
console.log(maxC);
elapsed=timer_end-timer_start;
console.log("Math.max(...;time:",elapsed);

console.log("Math.max.apply(");
timer_start = Date.now();

maxC=Math.max.apply(null,it);

timer_end = Date.now();
console.log(maxC);
elapsed=timer_end-timer_start;
console.log("Math.max.apply;time:",elapsed);
/* */

console.log("_.max");
timer_start = Date.now();
```

```
maxC=_.max(it);

timer_end = Date.now();
console.log(maxC);
elapsed=timer_end-timer_start;
console.log("_.max;time:",elapsed);

console.log("_.reduce");
timer_start = Date.now();

maxC=_.reduce(it,function(max,elem){
              if (elem>max) return elem
              else return max;
            },it[0]
          );

timer_end = Date.now();
console.log(maxC);
elapsed=timer_end-timer_start;
console.log("_.reduce;time:",elapsed);

console.log("lo.max");
timer_start = Date.now();

maxC=lo.max(it);

timer_end = Date.now();
console.log(maxC);
elapsed=timer_end-timer_start;
console.log("lo.max;time:",elapsed);

console.log("lo.reduce");
timer_start = Date.now();

maxC=lo.reduce(it,function(max,elem){
              if (elem>max) return elem
              else return max;
            },it[0]
          );

timer_end = Date.now();
console.log(maxC);
elapsed=timer_end-timer_start;
console.log("lo.reduce;time:",elapsed);
```

Maximum érték meghatározása objektumok listájából

```
const _ = require('underscore');
const lo = require('lodash');
const data = require('./obj_14.js');
let it = data.it;

console.log("Max");

var timer_start;
```

```
var maxC;
var timer_end;

console.log("For");
timer_start = Date.now();

maxC=it[0].c;
for (let i=1; i<it.length; i++)
{
    if (it[i].c>maxC) {
            maxC=it[i].c;
           }
}

timer_end = Date.now();
console.log(maxC);
elapsed=timer_end-timer_start;
console.log("For;time:",elapsed);

console.log("foreach");
timer_start = Date.now();

maxC=it[0].c;
it.forEach(elem => {
           if (elem.c>maxC) {
             maxC=elem.c;
           }
});

timer_end = Date.now();
console.log(maxC);
elapsed=timer_end-timer_start;
console.log("foreach;time:",elapsed);

console.log("reduce");
timer_start = Date.now();

maxC=it.reduce((max,elem) => {
            if (elem.c>max) return elem.c
            else return max;
           },
           it[0].c
);

timer_end = Date.now();
console.log(maxC);
elapsed=timer_end-timer_start;
console.log("reduce;time:",elapsed);

console.log("_.max");
timer_start = Date.now();

maxC=_.max(_.map(it,function(elem){return elem.c;}));

timer_end = Date.now();
```

```
console.log(maxC);
elapsed=timer_end-timer_start;
console.log("_.max;time:",elapsed);

console.log("_.map");
timer_start = Date.now();

_.map(it,function(elem){return elem.c;});

timer_end = Date.now();
elapsed=timer_end-timer_start;
console.log("_.map;time:",elapsed);

console.log("_.reduce");
timer_start = Date.now();

maxC=_.reduce(it,function(tmpMax,elem){
              if (elem.c>tmpMax) return elem.c
              else return tmpMax;
            },it[0].c
          );

timer_end = Date.now();
console.log(maxC);
elapsed=timer_end-timer_start;
console.log("_.reduce;time:",elapsed);

console.log("lo.max");

timer_start = Date.now();
maxC=lo.max(lo.map(it,function(elem){return elem.c;}));

timer_end = Date.now();
console.log(maxC);
elapsed=timer_end-timer_start;
console.log("lo.max;time:",elapsed);

console.log("lo.map");
timer_start = Date.now();

lo.map(it,function(elem){return elem.c;});

timer_end = Date.now();
elapsed=timer_end-timer_start;
console.log("lo.map;time:",elapsed);

console.log("lo.reduce");
timer_start = Date.now();

maxC=lo.reduce(it,function(tmpMax,elem){
              if (elem.c>tmpMax) return elem.c
              else return tmpMax;
            },it[0].c
          );
```

```
timer_end = Date.now();
console.log(maxC);
elapsed=timer_end-timer_start;
console.log("lo.reduce;time:",elapsed);
```

Maximum érték és nevek meghatározása

```
  const _ = require('underscore');
  const lo = require('lodash');
  const data = require('./obj_v2_1000000.js');
  let it = data.it;

  console.log("Max names");

  var timer_start;
  var maxC;
  var names=[];
  var timer_end;

  console.log("For x2");
  timer_start = Date.now();

  maxC=it[0].c;
  for (let i=1; i<it.length; i++)
  {
      if (it[i].c>maxC) {
              maxC=it[i].c;
            }
  }
  names=[];
  for (let i=0; i<it.length; i++)
  {
            if (it[i].c==maxC) {
              names.push(it[i].n);
            }
  }

  timer_end = Date.now();
  console.log(maxC);
  console.log(names.length);
  elapsed=timer_end-timer_start;
  console.log("For x2;time:",elapsed);

  console.log("ForEach x2");
  timer_start = Date.now();

  maxC=it[0].c;
  it.forEach(elem => {
            if (elem.c>maxC) {
              maxC=elem.c;
            }
  });
  names=[];
  it.forEach(elem => {
```

```
            if (elem.c==maxC) {
              names.push(elem.n);
            }
});

timer_end = Date.now();
console.log(maxC);
console.log(names.length);
elapsed=timer_end-timer_start;
console.log("ForEach x2;time:",elapsed);

console.log("_.max+_.filter");
timer_start = Date.now();

maxC=_.max(_.map(it,function(elem){return elem.c;}));
names=_.filter(it,function(elem){return elem.c==maxC;});

timer_end = Date.now();
console.log(maxC);
console.log(names.length);
elapsed=timer_end-timer_start;
console.log("_.max+_.filter;time:",elapsed);

console.log("_.map");
timer_start = Date.now();

_.map(it,function(elem){return elem.c;});

timer_end = Date.now();
elapsed=timer_end-timer_start;
console.log("_.map;time:",elapsed);

console.log("_.max+_.where");
timer_start = Date.now();

maxC=_.max(_.map(it,function(elem){return elem.c;}));
names=_.where(it,{c:maxC});

timer_end = Date.now();
console.log(maxC);
console.log(names.length);
elapsed=timer_end-timer_start;
console.log("_.max+_.where;time:",elapsed);

console.log("_.map");
timer_start = Date.now();

_.map(it,function(elem){return elem.c;});

timer_end = Date.now();
elapsed=timer_end-timer_start;
console.log("_.map;time:",elapsed);

console.log("lo.max+lo.filter");
timer_start = Date.now();
```

```
maxC=lo.max(lo.map(it,function(elem){return elem.c;}));
names=lo.filter(it,function(elem){return elem.c==maxC;});

timer_end = Date.now();
console.log(maxC);
console.log(names.length);
elapsed=timer_end-timer_start;
console.log("lo.max+lo.filter;time:",elapsed);

console.log("lo.map");
timer_start = Date.now();

lo.map(it,function(elem){return elem.c;});

timer_end = Date.now();
elapsed=timer_end-timer_start;
console.log("lo.map;time:",elapsed);

// More effective - build together
console.log("For");
timer_start = Date.now();

maxC=it[0].c;
names=[];
names.push(it[0].n);
for (let i=1; i<it.length; i++)
{
    if (it[i].c>maxC) {
            maxC=it[i].c;
            names=[];
            names.push(it[i].n);
          } else if (it[i].c==maxC) {
            names.push(it[i].n);
          }
}

timer_end = Date.now();
console.log(maxC);
console.log(names.length);
elapsed=timer_end-timer_start;
console.log("For;time:",elapsed);

console.log("foreach");
timer_start = Date.now();

maxC=it[0].c;
names=[];
//names.push(it[0].n);
it.forEach(elem => {
          if (elem.c>maxC) {
            maxC=elem.c;
            names=[];
            names.push(elem.n);
          } else if (elem.c==maxC) {
```

```
            names.push(elem.n);
          }
});

timer_end = Date.now();
console.log(maxC);
console.log(names.length);
elapsed=timer_end-timer_start;
console.log("foreach;time:",elapsed);

console.log("reduce");
timer_start = Date.now();

maxO=it.reduce((tmpMax,elem) => {
            if (elem.c>tmpMax.maxC) return
{maxC:elem.c,names:[elem.n]}
            else if (elem.c==tmpMax.maxC) { newnames=tmpMax.names;
newnames.push(elem.n); return {maxC:tmpMax.maxC,names:newnames};}
            else return tmpMax;
          },
          {maxC:it[0].c,names:[]}
);

timer_end = Date.now();
console.log(maxO.maxC);
console.log(maxO.names.length);
elapsed=timer_end-timer_start;
console.log("reduce;time:",elapsed);

console.log("_.reduce");
timer_start = Date.now();

maxO=_.reduce(it,function(tmpMax,elem){
              if (elem.c>tmpMax.maxC) return
{maxC:elem.c,names:[elem.n]}
              else if (elem.c==tmpMax.maxC) { newnames=tmpMax.names;
newnames.push(elem.n); return {maxC:tmpMax.maxC,names:newnames};}
              else return tmpMax;
            },{maxC:it[0].c,names:[]}
          );

timer_end = Date.now();
console.log(maxO.maxC);
console.log(maxO.names.length);
elapsed=timer_end-timer_start;
console.log("_.reduce;time:",elapsed);

console.log("lo.reduce");
timer_start = Date.now();

maxO=lo.reduce(it,function(tmpMax,elem){
              if (elem.c>tmpMax.maxC) return
{maxC:elem.c,names:[elem.n]}
              else if (elem.c==tmpMax.maxC) { newnames=tmpMax.names;
newnames.push(elem.n); return {maxC:tmpMax.maxC,names:newnames};}
```

```
              else return tmpMax;
            },{maxC:it[0].c,names:[]}
          );

timer_end = Date.now();
console.log(maxO.maxC);
console.log(maxO.names.length);
elapsed=timer_end-timer_start;
console.log("lo.reduce;time:",elapsed);
```

B. Haskell implementációk

Tesztadatok

Objv9.hs formátuma:

```
module Objv9 where
  v = [
     ("n354013",354013),
     ("n692809",692809),
     ("n238100",238100),
     ("n939538",939538),
     ("n242031",242031),
     ("n908242",908242),
     ("n283977",283977),
     ("n831996",831996),
     ("n650007",650007)];
```

Maximum érték és nevek meghatározása

```
import Control.Exception
import Data.Time
import Objv9 as Obj

maximum' :: Ord a => [a] -> a
maximum' = foldr1 (\x y ->if x >= y then x else y)

maximum'' :: Ord a => [a] -> a
maximum'' [x]       = x
maximum'' (x:x':xs) = maximum'' ((if x >= x' then x else x'):xs)

maximum''' :: Ord a => [a] -> a
maximum''' [x]       = x
maximum''' (x:x':xs) | x >= x'   = maximum''' (x:xs)
maximum''' (x:x':xs) | otherwise = maximum''' (x':xs)

values :: [Integer]
values = [v | (n,v) <- Obj.v ]

values' = map fieldValue Obj.v
    where
    fieldValue (n,v) = v

names = map fieldName Obj.v
    where
    fieldName (n,v) = n
```

```
--- maxNames :: Integer -> [String]
maxNames m' a = map fieldName (filter (flip equalValue m') a)
    where
    fieldName (n,v) = n
    --- equalValue :: (Integer -> a) -> Bool
    equalValue (n,v) m = (v==m)


main = do
    print Obj.v

    start <- getCurrentTime
    let m=maximum values
    print (m)
    end <- getCurrentTime
    print (diffUTCTime end start)

    start <- getCurrentTime
    let m=maximum' values
    print (m)
    end <- getCurrentTime
    print (diffUTCTime end start)

    start <- getCurrentTime
    let m=maximum'' values
    print (m)
    end <- getCurrentTime
    print (diffUTCTime end start)

    start <- getCurrentTime
    let m=maximum''' values
    print (m)
    end <- getCurrentTime
    print (diffUTCTime end start)


    start <- getCurrentTime
    let m=maximum values'
    print (m)
    end <- getCurrentTime
    print (diffUTCTime end start)

    start <- getCurrentTime
    let m=maximum' values'
    print (m)
    end <- getCurrentTime
    print (diffUTCTime end start)

    start <- getCurrentTime
    let m=maximum'' values'
    print (m)
    end <- getCurrentTime
    print (diffUTCTime end start)
```

```
start <- getCurrentTime
let m=maximum''' values'
print (m)
end <- getCurrentTime
print (diffUTCTime end start)

--- print names
start <- getCurrentTime
print (maxNames m Obj.v)
end <- getCurrentTime
print (diffUTCTime end start)
```