A ProgCont megújítása és a középiskolai tehetséggondozás

Aszalós László[1], Kósa Márk[2], Pánovics János[3]

{[1]aszalos.laszlo, [2]kosa.mark, [3]panovics.janos}@inf.unideb.hu
DE IK

Absztrakt. Az egyetemi képzés tömegképzésével szinte minden egyetemen megszületett egy saját fejlesztésű rendszer, mely többé-kevésbé automatikus módon, rendszerint input-output párokra építkezve ellenőrzi a programozási beadandókat. A Debreceni Egyetem fejlesztése a ProgCont névre hallgat, és 2011 óta használjuk órai számonkérésekre és versenyek szervezésére. Az organikusan fejlődő program mára elért egy olyan szintet, ahol már módszeres tervezéssel érdemes továbblépni, összegyűjtve a különféle igényeket. A következő programverzió moduláris felépítése lehetővé tenné, hogy az alapfunkciók megtartása mellett különféle igényeket más és más modulok segítségével valósítsunk meg. Ebben a cikkben a tervek bemutatása során a középiskolások oktatására vonatkozó elképzeléseinket domborítanánk ki, remélve, hogy ezzel minél szélesebb kört tudunk bevonni a tervezésbe.

Kulcsszavak: tehetséggondozás, önképzés, programozás

1. Bevezetés

Azáltal, hogy az informatika egyre több területre tör be az életünkben, egyre több szoftverre és vele együtt egyre több szoftverfejlesztőre van szükség. Míg a nyolcvanas években közel harminc diák kezdte el a programozó matematikus szakot évente, mára az Informatikai Karra évente közel 400 új hallgató iratkozik be. Az a harminc ember már kemény verseny után ért célba, jó elméleti alapokkal rendelkezett matematikából és fizikából, sőt szinte mindegyikük ismert valamilyen programnyelvet, így az alapvető feladat az ismeretek rendszerezése volt, mintsem az alapok megtanítása. Ahogy a képzés a kilencvenes évektől eltolódott a tömegképzés irányába, egyre több olyan hallgatóval találkoztunk, akik alapvető programozási ismeretek nélkül kezdték meg a tanulmányaikat, így a tematikában is egyre nagyobb arányban fordultak elő az alapismeretek. Viszont ez nem csupán az előadásokat változtatta meg, hanem a gyakorlatok is átalakultak. Míg korábban a gyakorlatvezető a jobbaknak kiadta a feladatot, és foglalkozhatott a néhány gyengébb képességű diákkal, mostanra az arányok felcserélődtek: csak néhány gyakorlott, programozási ismeretekkel rendelkező diák kerül be egy-egy csoportba, a túlnyomó többség kezdő programozásban.

Ezzel természetesen a személyre szabott foglalkozás lehetetlenné válik, mert nincs idő tucatnyi programkezdemény részletes átvizsgálására, elvi, stílusbeli hibák felkutatására, az alkalmazott módszer részletes átbeszélésére, amire még volt idő, mikor csak 2–3 hasonló program volt egy gyakorlati csoportban. Persze a diákok teljesítményét valahogy vizsgálni kell. Az általuk írt programokat fekete dobozként tekintve, a viselkedésüket figyelve, különféle inputokra adott válaszaik alapján lehet a véleményt meghozni.

Az ilyen fajta ellenőrzés természetesen automatizálható, egy program figyelheti, hogy az elkészült forráskódok lefordíthatóak-e (nincs bennük lexikai vagy szintaktikai hiba), a megadott tesztesetek lefutnak-e (nem fedezünk fel velük szemantikai hibákat), és a futási eredmények megfelelnek-e a feladat specifikációjának vagy sem. Miután volt olyan időszak, amikor közel 500 diák hallgatta a bevezető programozás kurzust, és hetente több programocskát kellett megírniuk, nagyon sürgető volt egy ilyen rendszer elkészítése. Ennek megfelelően erre készült el a rendszer, mely aztán a helyi igények mentén időről időre organikusan továbbfejlődött [1–4]. Egyik házi versenyünk által generált vadhajtásában az

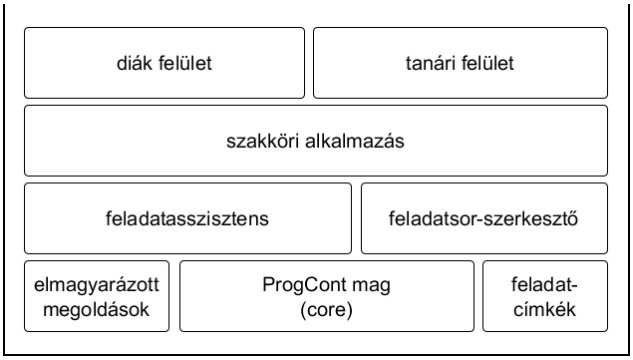
output nem volt egyértelműen meghatározható, így az egyszerű szöveg-összehasonlítás (a beadott és a standard program kimenetei) helyett feladatonként változó ellenőrprogramot kellett megírni és a feladathoz rendelni.

Jó programozói szokás szerint mindig a funkcionalitáson volt a hangsúly, de ez be is korlátozta a felhasználhatóságát, mert az adminisztrációs felület a mai napig pilótavizsgás. Miután egyre több megkeresés érkezik a rendszer karunkon kívüli felhasználására, így az egész rendszer újratervezése van folyamatban, és ehhez állítjuk össze jelenleg a követelmények rendszerét, bevonva a leendő felhasználókat.

A cikknek a felépítése a következő: a 2. fejezetben egy olyan modulszerkezetet mutatunk be, mely véleményünk szerint a középiskolai programozói szakkörök vagy differenciált informatikaórák szervezésében használható. Majd felvázolunk pár ötletet, miként lehet egyes modulok cseréjével különféle versenyeket rendezni, illetve hogyan alkalmazható a rendszer önképzésre. Végül összefoglaljuk elképzeléseinket.

2. A tervezett felépítés

Miután az elmúlt években felmerült igények elég szerteágazóak, a legésszerűbb a rendszert modulárisan felépíteni, és az elkészült modulokat akár több helyen is felhasználni. Az alábbi ábra egy lehetséges középiskolai szakköri rendszert és az azt kiszolgáló modulokat mutatja be:



1. ábra: Szakköri alkalmazás felépítése.

2.1. ProgCont mag

Az egész rendszer alapmodulja a feladatokat, a hozzá kapcsolódó tesztpárokat (input és output) és az esetleges időkorlátokat tartalmazó adatbázist, valamint a tesztelést lehetővé tevő és adminisztráló programkörnyezetet tartalmazza.

Miután az input és output néhány programozási nyelvnél nem a legegyszerűbb terület (gondoljunk csak a Haskell IO monadjára), így a programozás kezdeti lépéseinél a teljes program írása helyett csupán egy-egy függvény megírását is előírhatjuk. Ebben az esetben a függvényt körbevevő forrás részei is az adatbázisban kaphatnak helyet.

Ez a mag megkap egy szövegfájlt, egy egyedi azonosítót, feladatazonosítót, a használt programnyelv elnevezését. Ha a feladat egy függvény megírása volt, a rendszer összeállítja a teljes forrást, mielőtt lefordítaná, ha viszont programot kértünk, egyből fordíthatja a forrást a megadott nyelvhez tartozó fordítóval, és futtatja a kapott programot. Az őt hívó programnak visszaküldi az eredményeket, kiegészítve a megkapott azonosítókkal. Arról a magnak nincs tudomása, hogy ez most egy gyakorló feladat megoldása, dolgozat- vagy versenypélda volt, teszi a dolgát.

2.2. Címkeadatbázis

A mag adatbázisa mára már több ezer feladatot tartalmaz. Ezek között vannak könnyebbek, nehezebbek. Vannak feladatok angol nyelven és vannak magyarul is, sőt sok feladat mindkét nyelven megtalálható. Az elmúlt évek fejlesztése révén már több programozási nyelvet is lehet használni (C, C++, C#, Java, Pascal, Python), de ezek számát növelni kívánjuk a karunkon fakultatívan oktatott egyéb nyelvekkel, hogy azok begyakorlását is elősegítsük.

Ezeket a feladatokat címkékkel kívánjuk ellátni, hogy segítsük az eligazodást közöttük, azaz a feladatokat kereshetővé tegyük. Fontos lehet ez az önálló tanulásra kész diáknak (legyen az középiskolás vagy egyetemista), de a feladatsorokat összeállító tanárnak is, aki valamilyen tematika mentén kívánja a diákokat felkészíteni, vagy olyan feladatsort kíván összeállítani, melyben a legtöbb programozási módszer előfordul.

A címkézés kiterjedne a megoldás programnyelveire, a feladat nehézségére (a nyelv utasításkészletét begyakorló vagy nehéz matematikai ismereteket, adatszerkezeteket, módszereket felhasználó feladatok), az alkalmazni javallt adatszerkezetre, programozási módszerre (pl. dinamikus programozás).

A címkeadatbázis alapvető szerepe a feladatok között az előbbi szempontok szerinti keresés. Viszont nem tartjuk kizártnak, hogy ugyanazokhoz a feladatokhoz különböző szinteken (középiskola, felsőoktatás stb.) más és más címkeadatbázisokat érdemes használni.

2.3. Elmagyarázott megoldások

Múzeumokban gyakran látni, hogy kezdő festők a nagy mesterek műveit másolják, elsajátítva különféle speciális megoldásokat. Hasonló dolog létezik az informatikában, egy alaposan körüljárt feladatmegoldás után sokat lehet azzal tanulni, hogy valaki megmutatja/bemutatja az optimális/hivatalos megoldást. Tervünkben szerepel egy adatbázis elkészítése, melyben a gyakorló feladatokhoz tartozó, kisebb-nagyobb részletességgel bemutatott megoldások szerepelnének. Itt a megvalósítás során arra törekszünk, hogy segítségével más-más szinten lehessen elmagyarázni ugyanannak a feladatnak a megoldását egy kezdőnek, mint egy haladónak. Mivel az alkalmazott programnyelv korlátai vagy lehetőségei is befolyásolják a megoldást, így itt is elképzelhető több párhuzamos adatbázis létezése.

2.4. Feladatsor-szerkesztő

Véleményünk szerint a szakkörvezető informatikatanárnak hatalmas segítséget jelent, ha megnyílik előtte az általunk évek során bővített feladattár, és a címkeadatbázis felhasználásával ebből az igényei szerint csemegézhet. Ezek segítségével otthoni megoldásra szánt feladatsort állíthat össze, és a szakkörön már csak az iránymutatással, a hibás megoldások átbeszélésével, közös megoldásával kell foglalkoznia. Sőt a magyarázott megoldások segíthetnek a szakkörre felkészülésben, a régi ismeretek felelevenítésében vagy esetleges pótlásában. A feladatsorok elmenthetőek, évről évre újra kiadhatóak, sőt reményeink szerint az egyes tanárok között igény szerint megoszthatóak, hogy a szakkörtartás terheit lehetőség szerint minél nagyobb mértékben csökkentsük.

2.5. Feladatasszisztens

Kevés dolog rombolja az ember motiváltságát annál, mint amikor elakad egy számítógépes problémával, legyen az bármilyen természetű. A számítógépnek mindig igaza van, de nem képes elmagyarázni, hogy miért is. Az informatikatanár nem azért van, hogy éjjel-nappal a diákja rendelkezésére álljon, és ha a diák elakadt – mondjuk egy programozási feladattal –, azonnal segítsen. Bár évtizedek óta vannak önsegélyező közösségek (levelező listák, Usenet-, majd Google-csoportok, Stack Overflow és hasonló portálok), melyekhez hasonlót mi is be kívánunk indítani Debrecenben, ám itt sem feltétlenül kap perceken belül segítséget a diák. Gyakorló feladatok esetén érzésünk szerint nem szentségtörés egy olyan inputot a diák tudomására hozni, melyen megbukott a programja, sőt sokadik próbálkozásra a

hozzá tartozó outputot sem. Akár a feladathoz kapcsolódó magyarázat egy része is megosztható a diákkal, vagy mások megoldásai is.

Persze ezt nagyon finoman hangolhatóvá kell tenni. Programozói verseny esetén szó sem lehet ilyen előny biztosításáról. Néhány önképző portál, mint például a *4clojure* [5] vagy a *Project Euler* [6], a magyarázatokat vagy mások megoldását csak a feladatok megoldása után teszi elérhetővé, így kényszerítve ki az önálló munkát. Ahol hiányzik a megoldásokat ellenőrző portál, ott a közösség más módon talál utat: a kultikussá vált kötet feladatainak megoldásait a *SICP-Solutions* nevű wikiben [7] rendszerezték, ami időnként akár több évtizedes programozói gyakorlat után sem haszontalan.

Szakkör gyakorló feladatai esetén vagy önálló gyakorlás során a megoldás részletes megismerése tolerálható, de hogy mennyire, azt már a magasabb szinten elhelyezkedő, a következő alfejezetben bemutatandó modul határozza meg.

2.6. Szakkör alkalmazás

A helyi igényeknek megfelelően testre szabott modul fogja össze a tanárok és a diákok tevékenységeit, belépéshez kötve a helyi adatok, szolgáltatások elérését. Ez a modul gyűjtheti a tagok helyes és hibás feladatmegoldásait, ezek száma alapján bónusz-malusz rendszert lehet kialakítani: a jó megoldásokért pontok járnak (a rosszakért meg esetleg levonások), melyekkel szükség esetén meg lehet venni a feladatasszisztens segítségét, vagy bele lehet nézni a szakkör többi diákjának megoldásába. A pontszámokkal helyi dicsőségtáblát lehet kialakítani – ami akár belépés nélkül is elérhető –, és még sok minden kitalálható, helyileg más és más modul-továbbfejlesztésekkel.

2.7. Diák felület

Az előző modul lehetőségeit két grafikus felület használja fel. Ezek megvalósíthatóak webes alapokon, de akár mobilalkalmazás is készíthető ugyanerre. Ezek a felületek akár helyi specialitásokat is tartalmazhatnak, helyet adva a diákok kreativitásának.

A beléptetés miatt minden diák csak a rá vonatkozó feladatokkal találkozik, ezen töltheti fel a megoldásait, itt kapja vissza az eredményeket, illetve igényelheti a feladatasszisztens által lehetővé tett segítségeket, továbbá a tanára által megosztott (jó vagy rossz) megoldásokat is itt érheti el.

2.8. Tanár felület

Ez a felület hasonló a diákokéhoz, csak az elérhető funkcionalitások mások. Lehetőség van feladatsorok összeállítására, az azokra beadott megoldásokhoz tartozó statisztikák generálására, ezáltal egyedi megoldások kiválasztására (ki volt az, aki ezt a feladatot megoldotta, vagy ki az, aki nem; mit nem sikerült még Steinmannak sem megoldania), illetve saját megoldás beküldésére, ha ezt kívánja bemutatni/megosztani az oktató.

3. Alkalmazásvariánsok

Napjainkban a tanár talán legnagyobb ellensége a motiváció hiánya. A hetvenes-nyolcvanas években a széles tömegek lehetséges felemelkedési módját a diploma megszerzése jelenthette. Viszont a férőhelyek száma igen limitált volt, ezért éles verseny volt az egyetemre készülők között, próbáltak minél jobb jegyeket szerezni, minél jobb helyezést elérni a különféle tanulmányi versenyeken.

Napjainkban, mikor minden második fiatal bekerül az egyetemre, nem szükséges ennyire küzdeni, de mivel a versenyszellem minden ember sajátja, érdemes erre építeni még az informatikában is, ezzel megtartva a diákok motiváltságát. Ezekhez természetesen újabb modulok fejlesztése szükséges, de részben lehet építeni a már bemutatott modulokra is.

3.1. Helyi programozói versenyek

A regionális programozói versenyünk, vagy az ACM mintájára lehet helyi, megyei versenyeket rendezni. A versenyszabályok (a jó megoldásért járó jutalmak, a rossz megoldásokért járó levonások, a feladatok beküldésének módja, a végleges sorrend, a kifele kommunikált állás) a versenymodulban kerül implementálásra. Itt a beküldött megoldás egyből kiértékelésre kerül. A megoldás futása rendszerint másodperceket jelent, és a verseny időtartama is csak pár óra.

3.2. Levelező verseny

Az előző kategóriában a feszes időkorlát miatt nem feltétlenül szépségdíjas megoldások születnek, melyeket később évekig mutogatnak a szakkörökön. Tegyük a szívünkre a kezünket, a legtöbb cégnél is határidőre dolgoznak, ott sem a szépség vagy a hatékonyság az elsődleges szempont, hanem hogy fusson az a program és teljesítse a specifikációt.

Épp ezért van helye egy olyan versenynek, ahol nemcsak összecsapni kell a megoldást, hanem dédelgetni is egy kicsit. A szépség nem objektív kategória, így a hatékonyság adja meg a végső sorrendet ebben az esetben. Annak érdekében, hogy a versenyszervezők munkája egyszerűsödjön, néhány közel triviális tesztesettel minden versenyző kipróbálhatja a megoldását, és az ezen a szűrőn átment megoldások eltárolódnak. A beküldési határidő (ami akár több napos, hetes is lehet) után már komolyabb feladatokkal találkoznak az eltárolt programok, és amelyikük a leggyorsabban birkózott meg velük, az a győztes.

Itt akár több perces futási idők is engedélyezettek, és nincs korlátozva a külső segítség, a szakirodalom használata. Ekkor el lehet mélyedni egy adott témakörben és megkeresni a leghatékonyabb megoldási módszert, a leggyorsabb implementációt.

3.3. Rókavadászat

A tesztekkel történő megoldásvizsgálat esetén a diákot a programírásra, az oktatót (egység)tesztek írására/generálására korlátozzuk. Szakesteken, fordított napokon ezek a szerepek felcserélhetőek. Mint ahogy ilyen napokon a diák oktat, a tanár pedig a padban rosszalkodik, hasonló szellemű programozási verseny is rendezhető. Itt az informatikatanár az, akinek programot kell írnia adott feladatra. De hogy a diák is szerepet kapjon, neki kell az input-output párokat megadni. Ezek a tesztadatok egy előzetes ellenőrzésen mennek keresztül egy etalon (hivatalos) megoldás segítségével, és mondjuk hibás tesztek beadása esetén az érintett diák kiesik. Ha sikerül elkapnia a diáknak a tanárt, azaz hibát találnia a programjában az általa írt teszttel, akkor azért pontot kap. Adott pontszám elérése a cél, aki azt legkorábban eléri, az a győztes. Lehet nehezíteni a versenyzők feladatát: egyre rövidebb időt megadni a tesztek, illetve programok megírására. Itt a program és vele együtt a teszt megírására szánt idő akár 1-2 percre is korlátozható magasabb szinteken, vagy csak addig lehet tesztet írni, míg a tanár a programját írja. A tesztek természetesen generálhatóak egy, a forduló során megírt program révén is (itt lehet bevezetni a diákokat az automatizált tesztelésbe). Akár több tanár is versenyezhet, ilyenkor az győz közülük, akit utoljára sikerül elkapni. A korábbiaktól eltérően ekkor a rendszer magjának egy variánsát kell használni, ugyanis nem a tárolt teszteseteket ellenőrzi a mag, hanem a diákok beküldéseit.

3.4. Távszakkör

Természetesen van olyan eset, mikor nincs meg a kritikus tömeg, és nem indul be az informatika szakkör az iskolában, vagy nincs megfelelő képességű informatikatanár. Ekkor a motivált diák alapvetően magára marad. Annak érdekében, hogy őt se veszítsük el, a karunkon elindítunk egy informatikai szakkört. A debreceni helyszín miatt alapvetően a vonzáskörzet diákjaira számítunk, de ezeket a szakköri alkalmakat rögzítenénk és online elérhetővé kívánjuk tenni. Egy-egy alkalomhoz egy-egy feladatsor készül, mely egy egyszerű regisztráció után elérhető. A soron következő szakkör időpontjában a kidolgozott megoldások is elérhetővé válnak (országhatáron belül és kívül is).

Persze ez a szakkör csak annak érdekes, aki már egy jó szinten tud programozni. Viszont próbáljuk önállóan elérhetővé tenni ezt a szintet is. Egyik EFOP projektünk keretében egy Python tankönyvet magyarra fordítunk és szabadon elolvashatóvá, letölthetővé teszünk, így innen az alapok megtanulhatóak, és a kezdők számára kidolgozott feladatsorainkat (valamint a megoldásaikat) a ProgCont rendszeren belül szintén elérhetővé tesszük, s reméljük, hogy ezzel minden motivált érdeklődő számára megnyitjuk a programozás csodálatos világát.

3.5. Önálló felkészülés

Végül ne hagyjuk ki azt az esetet sem, amikor a diák – a korára jellemző – lázadó módon eltérne a programozási nyelvi kánontól, és mondjuk egy Youtube-videó hatására fejébe veszi, hogy Lua, Go vagy éppen Elixir nyelvet akar tanulni. Ekkor hiába van számára elérhető módon egy szakkör, melyen a C++-t meg a Javát használják, a speciális igényeiben ez nem sokat segít. De nemcsak a diákok lehetnének aktív használói az önálló felkészülést támogató rendszernek, hanem mindazok, akik valami újjal szeretnének megismerkedni. Jelen sorok írói belső késztetést éreznek, hogy pár évente megismerjenek egy új programozási nyelvet, gyakran függetlenül az ipari igényektől, csak saját szórakozásra. Ilyen esetekben lényegében az új nyelv programozási környezetét kell a rendszer magjához illeszteni, és már használatra kész, lehet ilyen nyelven megoldásokat beküldeni.

A felhasználó választhatja a már elkészült (tematikus) feladatsorokat, vagy választhat egyedi feladatot is: a címkéket tartalmazó adatbázist kezelő webes alkalmazás megfelelő paraméterek kiválasztásával feldobja az illeszkedő feladatokat (vagy csak azoknak egy részét). Reszponzív dizájnt alkalmazva ez a feladatkereső alkalmazás mobiltelefonról is elérhető, így a diák/érdeklődő a busz- vagy vonatállomáson kiválaszthat pár feladatot, melyen hazáig töprenghet, s majd otthon megoldhatja azt.

Lehetővé kívánjuk tenni a felhasználó számára az adatbázisból kiválasztott és általa jól megoldott feladatok elmagyarázását, így ő akár az adott nyelv pionírjává válhat a rendszerben, ha nemcsak élvezi a rendszer nyújtotta előnyöket, hanem önzetlenül hozzá is járul annak fejlesztéséhez.

4. Konklúzió

A tömegképzés hátrányaiból próbálunk előnyt kovácsolni. Az ismétlődő, kreativitást nem igénylő, automatizálható programellenőrzést – sok egyetemhez hasonlóan mi is – egy számítógépes rendszerre bíztuk, s így a kontaktórákat hasznosabban töltheti a tanár és diák. Az évek során az igényeknek megfelelően organikusan fejlődő rendszerünk hamarosan megújul. A középiskolákat is érintő terveinket mutattuk be ebben a cikkben, remélve, hogy ez alapján elindul egy párbeszéd, és minden érintett tanár véleményét felhasználva elkészíthetünk egy követelményjegyzéket, mely vezérfonala lesz a fejlesztésünknek, és ezáltal egy, a középiskolákban is jól használható rendszert sikerül kialakítanunk, mellyel a diákok tudásszintje emelhető, és így mi is képzettebb diákokat tudunk felvenni évek múlva.

Irodalom

1. *ProgCont feladatkiértékelő rendszer*, https://progcont.hu (utoljára megtekintve: 2017.11.03.)
2. Kádek Tamás, Kósa Márk, Pánovics János: *Programozó versenyek támogatása webes alkalmazással*, In: Biró Károly-Ágoston, Sebestyén-Pál György (szerk.): ENELKO 2011 SZÁMOKT2011: XII Nemzetközi Energetika-Elektrotechnika Konferencia; XXI Nemzetközi Számítástechnika és Oktatás Konferencia. 318 p. Konferencia helye, ideje: Kolozsvár, Románia, 2011.10.06–2011.10.09. Kolozsvár: Erdélyi Magyar Műszaki Tudományos Társaság (EMT), 2011. pp. 184–187.
3. Kádek Tamás, Kósa Márk, Pánovics János: *A ProgCont szoftverrel támogatott programozó versenyek tapasztalatai*, In: Veronika Stoffová (szerk.): New Technologies in Science, Research and Education. Konferencia helye, ideje: Komárno, Szlovákia, 2012.09.10–2012.09.13. Komárno: Janos Selye University, 2012. pp. 152–157.

4. Kádek Tamás, Kósa Márk, Pánovics János: *Programozási ismeretek mérése a ProgCont rendszerrel*, In: Biró Károly-Ágoston, Sebestyén-Pál György (szerk.): XV. Nemzetközi Energetika-Elektrotechnika konferencia, ENELKO 2014; XXIV. Számítástechnika és Oktatás Konferencia – SzámOkt 2014. Konferencia helye, ideje: Székelyudvarhely, Románia, 2014.10.09–2014.10.12. Cluj-Napoca: Erdélyi Magyar Műszaki Tudományos Társaság (EMT), 2014. pp. 188–191.
5. *4Clojure*, http://www.4clojure.com (utoljára megtekintve: 2017.11.03.)
6. *Project Euler*, https://projecteuler.net (utoljára megtekintve: 2017.11.03.)
7. *SICP-Solutions*, http://community.schemewiki.org/?SICP-Solutions (utoljára megtekintve: 2017.11.03.)