SZAKDOLGOZAT

Sarkadi Sándor

Debrecen

2011

Debreceni Egyetem

Informatikai Kar

PARADOXONOK AZ INFORMATIKÁBAN

Témavezető:

Dr. Fazekas Gábor

Ph.D., egyetemi docens

Készítette:

Sarkadi Sándor

Programtervező informatikus

Debrecen

2011

Tartalomjegyzék

BEVEZETŐ

A mindennapoktól egészen a tudományos területekig találkozhatunk paradoxonokkal. Ezek általában valamilyen látszólagos ellentmondások, lehetetlenségek, amik bizony sokszor megnehezítik, és akár le is állíthatnak hosszabb időre valamilyen kutatást, attól függetlenül, hogy milyen szakterületen találkozunk velük. Ezeket az ellentmondásokat fontos ismernünk, mert így ha egy már definiált paradoxonnal találkozunk munkánk során, nem követjük el mi is ezekbe a hibákba, ezzel időt és pénzt spórolhatunk.

Ezért tartottam fontosnak, hogy szakdolgozatom fő témája az informatikában előforduló paradoxonok legyenek. Tájékozódásom és kutatásom alatt sajnos arra is rá kellett jönnöm, hogy kevés magyar nyelvű forrás található a szakirodalomban ezekről a jelenségekről, így külön megtiszteltetésnek érezhettem, hogy elsőnek én fordíthattam le ezeket a forrásokat anyanyelvünkre. Ezzel is remélem segítem és támogatom azokat, akik régóta dolgoznak ugyanebben a témában, illetve felkeltsem azok érdeklődését is, akik eddig nem fektettek elég nagy hangsúlyt az informatika legnagyobb ellentmondásaiba.

Az informatikának lehetetlenségeinek három nagyobb csoportját különböztetjük meg. Az első, a közgazdasági tudományokban is jártas szakértők által ismert Solow-paradoxon. Ez a jelenség arra mutat rá, hogyan állhat elő az a látszólag érthetetlen ellentmondás, miszerint a haladással nem haladni fogunk, hanem a kívánt eredménytől ellentétben visszaesés, rosszabb esetben hanyatlás következik be egy adott folyamatban. A második és harmadik témakör szorosan kapcsolódnak egymáshoz, ezek a homonkulusz-paradoxon és a mesterséges intelligenciákhoz kapcsolódó vezérlőegység paradoxonok. A homonkulusz-paradoxon szerint ugyanis egy folyamatot nem az a réteg, befogadó egység dolgoz fel először, aminek gondoljuk, hanem egy másik, létező entitás, egy „homonkulusz” (amit pici emberkének is szoktak nevezni a szakirodalomban) hajtha végre. A vezérlőegység ehhez nagyon hasonló, de mégis különbözik abban, hogy itt nem egy értelmes, élő entitás hozza meg a döntéseket egy hétköznapi ember szerint alapul vett réget, interfész helyett, hanem

Informatikus lévén tudom, milyen fontos, hogy elméletünket példákkal, jelen esetben kódokkal támasszuk alá a jobb megérthetőség és követhetőség kedvéért. Ezért én egy nagyon egyszerű, minimális funkciókkal működő, PHP nyelven megírt WebShop-ot hoztam létre, amin ezeket a tüneteket (jó ellenpéldát szolgáltatva) fogom bemutatni. Ezért a paradoxonok

elméleti háttere mellett fontosnak éreztem, hogy a PHP5-ös nyelv egy-két fontosabb függvényére, funkciójára és jellemzőjére is kitérjek, melyeket érthető és átlátható kódrészletekkel prezentálok.

A három nagy témakör előtt szerepelni fog kötelező jelleggel egy bevezető, amely a paradoxonokkal mint általános értelemben foglalkozik, amiben az általános, minden tudományágban megjelenő fogalom után konkretizálódunk az informatikai paradoxonokra. Ennek legismertebb, bár nem tisztán informatikai vonatkoztatású ellentmondása, a Solow-paradoxon nagyon sok közgazdasági vonatkoztatással bír, ezek szerves részei az eddig elkészült angol szakfordításomnak, amit majd saját gondolatokkal, meglátásokkal kívánok kiegészíteni.

Célom a témám bemutatása oly módon, hogy a szükséges szakszavakat minden használat előtt definiálni fogom, illetve minden kódrészletet megjegyzésekkel fogok ellátni a könnyebb megérthetőség és követhetőség érdekében. A megértés könnyítését szolgálják továbbá a három nagy témához csatolt példák. Ezek a példák az adott témakörhöz szorosan kapcsolódó aktuális példák és kódrészletek lesznek, de csak ott lesznek csatolva, ahol elengedhetetlenül szükségesek.

A Solow-paradoxonhoz remek példa a nemrég interneten megjelent hírek összessége, amely az Oroszországban és más európai országokban bevezetésre szánt, majd megbukott nyílt forráskódú operációs rendszerekről szólt. A Homonkulusz-paradoxonhoz és a vezérlő egység paradoxonhoz pedig egy egyszerű, de látványos php-kódban írt, leegyszerűsített webshopot fogok készíteni és bemutatni, amelyben direkt módon kiemelem a súgó és egyéb felhasználói segítség szerepét, és így tökéletes példaként fog szolgálni a paradoxon kiszűrésére. Végül a mesterséges intelligencia programokhoz kötődő egység ellentmondás bemutatásához szintén a php kódban fogok példát készíteni, de itt a Homonkulusz-paradoxonnal ellentétben ellenpéldát fogok mutatni.

1.Az informatikai és más tudományos paradoxonok kapcsolata

Gregory J. Chaitin egy 1947-es születésű argentín-ametikai matematikus és informatikus volt. Egyik híres tanulmánya megjelent a 2002-es American Scientist folyóiratban, a Computers, Paradoxes and the Foundations of Mathematics. Ez a dokumentum kiválóan bemutatja, hogy igenis léteztek olyan 20. századi nagy gondolkodók, akik még a legszigorúbb matematikában is találtak bőségesen ellentmondásokat, és képtelen

levezetéseket, és a véletlenszerűségeket.

Mindenki számára ismeretes, hogy a számítógép korunk egyik leghasznosabb eszköze. Sőt, a számítógépek szolgálnak a modern társadalom alapjául. Ennek ellenére az informatikai szakemberek többsége nem emlékszik arra, hogy a számítógépet eredetileg azért találták fel, hogy egy matematikai kutatások során előállt filozófiai vitát oldjon meg. Meglepően hangzik, de igaz.

A történetünk David Hirberttel kezdődött, a nagy hírű német matematikussal, aki a 20.század hajnalán kezdeményezte, hogy minden matematikai levezetést, magyarázatot teljes mértékben le kellene formalizálni. Kiderült azonban, hogy erre akkoriban senki sem volt képes, ezért az ötlete hatalmas bukásra volt ítélve. Más szempontból nézve azonban Hirbert ötlete sikeres volt, ugyanis később a formalizáció vált a 20. század legnagyobb úttörőjévé. Nem csak a matematikai levezetések megkönnyítését tette lehetővé, de a programozásban, a számítások elvégzésében és a bonyolult, időigényes műveletek meggyorsításában és leegyszerűsítésében is. Ez a történelemnek egy elfeledett része.

A történelem ennek a részét most úgy kívánjuk feleleveníteni, hogy nem mélyülünk el túlságosan a matematikai részletekben. Lehetetlen lenne ugyanis kitérni részletesen Bertrand Russell, Kurt Gödel és Alan Turing összes felfedezésére. Habár a türelmesebbek kedvet kaphatnak arra, hogy utánanézve és mélyreható kutatásokat végezve felfedezzék a tényleges szépségét, és ez a gondolatmenet lehetőleg ösztönözni fogja őket arra is, hogy megalkossák a saját matematikai véletlenszerűségen alapuló ötleteiket, meglátásaikat.

A Russell-féle Logikai Paradoxon

Kezdjük hát azzal a matematikussal, aki később filozófus, majd végül humanistaként fejezte be életútját, név szerint Bertrand Russellel. Russell szerepe kulcsfontosságú, ugyanis jó pár paradoxont vett észre a logikában is, amelyek elég megrázóak. Ezek mind olyan esetek, ahol az ésszerű következtetésekkel juthatunk el valamilyen ellentmondásba. Russel minden eszközével terjesztette ezeket a nézeteket, melyek komoly problémákat okozhatnak, és megoldásra vártak.

Ezek a paradoxonok, melyeket ő fedezett fel, nagy körben elterjedtek a matematikusok között, és hatalmas figyelmet keltettek. Mindezek ellenére összesen egyetlen egy ellentmondás kapta a nevét Russellről. Hogy megértsük a Russell-paradoxont, képzeljünk magunk elé olyan matematikai halmazokat, amik nem önmaguk elemei. És tegyük fel a kérdést, ez a halmaz eleme önmagának? Ha eleme önmagának, akkor ez nem lehet igaz, és

fordítva.

A halmazok halmaza a Russell-paradoxonban legjobban egy kis, távoli városkában lévő fodrászathoz hasonlítható, ahol a fodrász mindenkit megborotvál, akik nem borotválkoznak otthon maguktól. A leírás alapján ésszerűnek tűnik, amíg fel nem tesszük a következő kérdést, miszerint a fodrász megborotválja magát? Ugyanis csak akkor teheti ezt meg, ha ő magát nem borotválja sohasem. Most biztos azt gondolja mindenki, ki törődik ezzel a valótlan, kitalált borbéllyal? Első ránézésre a probléma nem más, mint szavak bugyuta, egyszerű játéka. De mikor a matematikai halmazok felépítésénél jön elő, nem tudjuk ilyen könnyen figyelmen kívül hagyni ezt a valós problémát.

A Russell-féle paradoxon nagyon hasonló egy ókori görög matematikai ellentmondásnak, melyet a mai világ többnyire Epimenidész-paradoxonnak is hív. A probléma gyökere abban rejlik, hogy felteszünk egy állítást, ami kimondja magáról, hogy hamis. Tényleg hamis? Ha az állítás hamis, akkor igaznak kell lennie. De ha igaz, akkor hamisnak kell lennie. Így bármely logikai értéket is rendeljük a mondat igazságtartalmához, gondban leszünk. Ennek létezik két részes változata is. Az első állítás kimondja, hogy a következő állítás igaz, míg a második állítás szerint az előző állítás hamis. Külön-külön nincs probléma velük, de kombinálva nem kaphatunk értelmes megoldást. Ahogy a borbélyos példát, ezt sem lehet mai ember szemével komolyan venni, mint bármely más logikai játékos fejtörőt, de a 20. században a nagy elmék véresen komolyan vették ezeket.

Hilbert megpróbálta kikerülni a formalizálás folyamatát, ez volt az egyik reagálás a problémára, ami kiköszörülheti a bajt. Azaz ha egy jónak tűnő kijelentés értelmezésében gond adódik, a megoldás szimbolikus logikát használva létrehoz egy mesterséges nyelvet, ami nagyon körültekintően írja le az ellentmondást elkerülő szabályokat. Sajnos ez a gyakorlati életben nem volt megoldható, ugyanis a mindennapi nyelv nem egzakt, szavai többféleképpen is értelmezhetőek, nem tudjuk, melyik szóalakhoz mely jelentés fog tartozni.

Hilbert menekülési terve

Hilbert létre akart hozni egy mesterséges értelmező nyelvet, amely matematikai levezetésekre képes. Ezért nagy figyelmet fektetett az axiómákon alapuló függvényekre, amelyek segítségével egy jól előre definiált (axióma) halmazban lévő szabályokkal a matematikai levezetéseket tényleges elméleti tételekké deriválta. A matematikával szembeni ily módú gondolkodás is szintén jelen volt már az ókori Görögországban, a szépen letisztult szabályokkal rendelkező Euklideszi matematikai rendszerben.

Más szavakkal, Hilbert célja nem volt más, minthogy nagyon precíz definíciók, triviális fogalmak és nyelvtani helyességet ellenőrző szabályokkal mindenki számára érthetően bemutassa, hogy működik matematika. Gyakorlatban ez temérdek munkát igényelt volna, így soha nem valósult meg, de elméletileg nem elhanyagolható ennek a lehetősége.

Hilbert javaslata mindezek ellenére egyenes, és könnyen érthető volt. Mégse számít forradalmi újításnak, hiszen csak a matematika történelmi vonalán haladt végig ismét, Leibniz, Boole és Frege elméleteit újra felhasználva. Célja bár az volt, hogy bármi módon képes legyen leformalizálni a matematika teljes tudományát, mindenki nagy meglepődésére azonban kiderült, ez lehetetlen vállalkozás lenne. Hilbert tévedett, de munkája nem veszett kárba, később erre az összegyűjtött tudásra már építkezni lehetett, hiszen maga a kérdés nagyon jó volt. Sőt, ezzel az egyetlen kérdéssel egy külön tudományágat nyitott a matematika területén belül, ami a *metamatematika* nevet kapta. A metamatematika egy önelemző területévé vált, ami azzal foglalkozik, mi az, amire a matematika képes, és mi az, amire nem.

Az alap elgondolás tehát az lett, hogy hiába vetettük el egy Hilbert-féle mesterséges nyelv ötletét, és hagytuk meg a teljesen kidolgozott formális axiómarendszert, akkor körübelül annyit értünk el vele, mintha egy papírra vonalak húzogatásával akarnánk teóriákat levezetni axiómákból. Bár az elgondolás jó volt, a matematikának tényleges jelentéssel kell bírnia. Ennek ellenére ha matematikát akarunk tanulni matematikai eszközökkel, le kell tisztázni mindennek a jelentését, és meg kell vizsgálnunk a mesterséges nyelvünket, minden tekintetben precíz szabályokkal, hogy minden kérdésre pontos választ adhassunk.

Ilyen kérdés lehet akár az is, hogy 1 egyenlő-e nullával (reményeink szerint nem). Tehát bármely utasítás esetén, amit nevezzünk A-nak, el kell tudnunk dönteni, hogy A teljesül, vagy nem teljesül a szabályaink alapján. A formális axiómarendszerünk akkor mondható csak teljesnek, ha vannak eszközeink mind A teljesülésének és nem teljesülésének bizonyítására is.

Hilbert lelki szemeivel már látta azoknak a gépi utasítási szabályoknak az elkészítését, amelyek majd oly pontosak lesznek, hogy bármely kérdéses tényről teljes bizonysággal el tudja majd dönteni, hogy az adott feladat betartja-e a szabályokat, vagy nem. Felismeri majd például az olyan hibákat is, amikor mondjuk a negyedik sorban szintaktikai hiba található, vagy valami a negyedik sorban, aminek a harmadikat kéne követnie, de mégsem követi. Hilbert elméletének betetőzése ez volt, nem több.

Az ötletének megvalósítását nem azt jelentette, hogy ezentúl csak így fogalmazhatunk majd matematikát meg, hanem erre lefordítva más is könnyen megérthesse a tudomány erősségeit. Hilbert erősen hitt abban, hogy képes lesz ezt megvalósítani, így képzelhető, mekkora sokként érte, mikor 1931-ben egy osztrák matematikus, név szerint Kurt Gödel megmutatta Hilbertnek, hogy a menekülési terve nem volt teljesen elfogadható, sőt, még elviekben sem megvalósítható.

Gödel hiányossági bizonyítása

Gödel földbe tiporta Hilbert elméletét 1931-ben, pedig még csak a bécsi egyetemi hallgatója volt. Később a brno-i származású matematikus Einsteinnel dolgozott a Princetonon.

Gödel szerint ugyanis Hilbert alapjaiban véve hibádzott. Valójában nincs rá semmilyen lehetőség, hogy a matematika egészére nézve kristálytisztán érthető formális axiómarendszert építsünk fel, ahol mindig el tudjuk dönteni, mi jó és mi rossz. Szerinte az elmélet már az elemi aritmetika terén ki is fullad, a 0,1,2,3... számsorozattal, az összeadás és szorzás műveleteinek hozzáadásával.

Bármely formális rendszer, ami megpróbálja magába foglalni csupán csak erre a két alapműveletre és a számsorozatra vonatkozó összes szabályát, sohasem lesz teljes. Pontosabban, vagy nem lesz teljes, vagy nem lesz helyes. Szóval, ha azt hisszük, lefedi a valóságot, nem fedi le a teljes valóságot. Különösen akkor, ha azt állítjuk, minden axióma és szabály levezetés ki tudja szűrni a hamis elméleteket, akkor biztosan lesznek olyan elméletek, aminek igazát nem tudjuk majd bizton megállapítani.

Gödel hiányossági bizonyítása nagyon jól meggondolt és ésszerű, egyben ellentmondásos is, itt-ott már az őrület és épelméjűség határainak vékony vonalán táncol. Gödel a paradoxon hatását egy hazugságon mutatja be, miszerint valaki azt állítja, hazudik, ami sem igaz, sem hamis nem lehet. Az osztrák matematikus így konstruált egy állítást, ami önmagát bebizonyíthatatlanná teszi. Ilyen konstrukciókat persze nagyon nehéz az

aritmetikai, matematikai állításokban készíteni, de ha nagyon ügyesek vagyunk, és képesek vagyunk rá, akkor könnyen belátható, hogy hamar gondban leszünk. Miért? Mert ha az állításról nem tudjuk bebizonyítani, igaz-e, vagy hamis, akkor a szabályok alapján hamisnak vesszük, és rossz eredményt fogunk kapni. Ha bebizonyíthatatlan, és ennek ellenére igaz, akkor pedig a matematika szabályrendszere hiányossá válik.

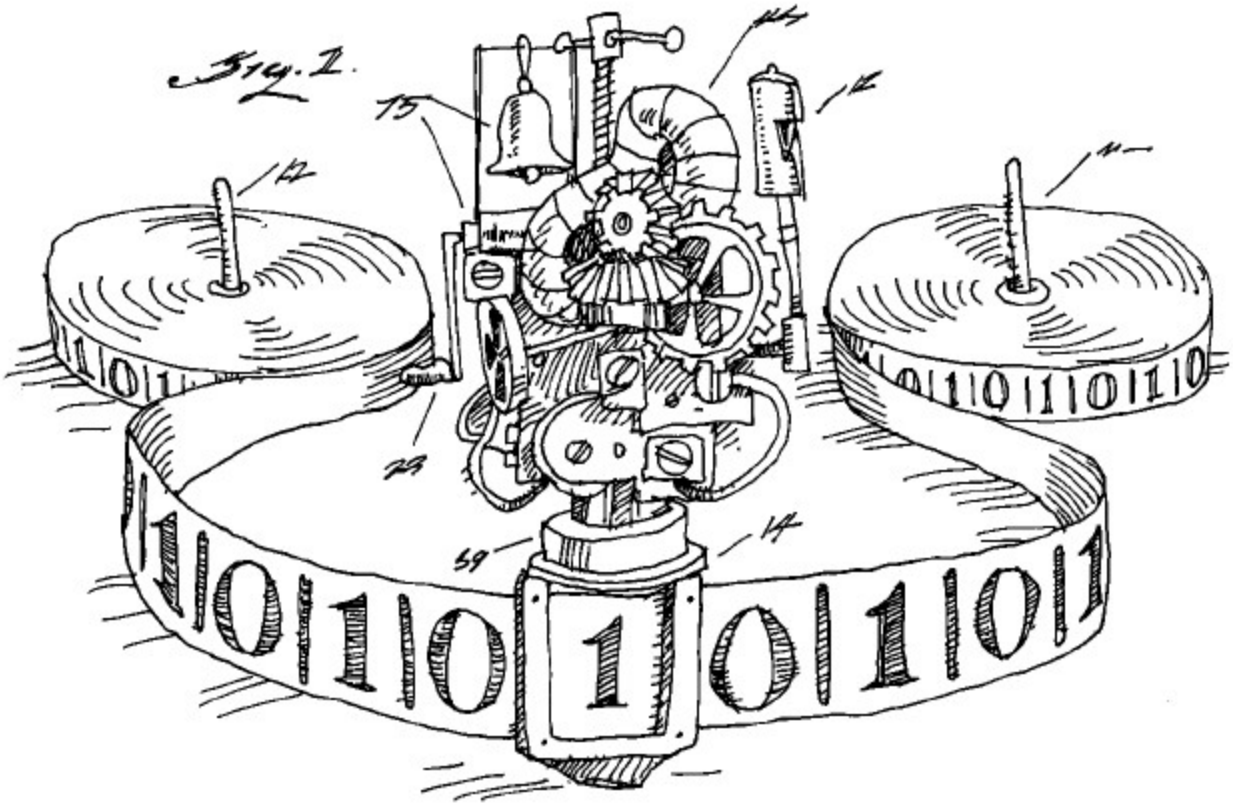
Gödel bizonyítása rengeteg bonyolult technikai adattal van alátámasztva. De ha a kezdeti jegyzeteit tekintjük, hamar ráismerhetünk a LISP programozási nyelv alapjaira. Ennek az oka, hogy Gödel bizonyítása rengeteg függvényt tartalmazott magában rekurzívan, listakezelő függvényekkel is, melyekre alapszik a LISP nyelv is. Így hiába nem voltak még számítógépek és programozási nyelvek 1931-ben, a mai utólagos tudásunkkal beláthatjuk, hogy egy programozási nyelv magja nem máshol született meg, mint Göring jegyzetének lapjain.

Egy másik ismert korabeli matematikus, Neumann János (aki mellékesen magyar volt, és az Amerikai Egyesül Államokban a számítógépes technológiák fejlesztésében teljesített kimagaslóan jelentős tevékenységet) azonnal elismerte Gödel észrevételeit. Érdekes tény, hogy Neumann János sohasem ismerte volna meg Gödel elméleteit, ha Gödel csak intelligens lett volna, de nem lett volna benne elég bátorság, hogy kiálljon Hilberttel szemben az igaza mellett.

Az emberek többsége szerint Gödel következtetése elsöprő hatású volt, és teljesen padlóra tette az addigi hagyományos matematikai filozófiát. Sajnos 1931-ben Európában másféle gondok ütötték fel a fejüket, ugyanis közelgett vészesen a világháború szele.

Turing gépe

A következő jelentősebb lépés előre öt év múlva történt meg Angliában, mikor Alan Turing felfedezte a „kiszámíthatóságtalanság” fogalmát. Emlékezzünk, mit mondott Hilbert a gépi utasítással elvégezhető folyamatokról, ami eldönti, az állítás teljesíti-e, vagy nem a szabályokat. Hilbert azonban sohasem tisztázta le, mit ért pontosan a gépi utasítás alatt, elmondása szerint valami, ami alatt tényleg egy gépet értünk (egy olyan gép, amit ma már Turing-gépnek nevezünk).



A fenti ábrán látható a Turing-gép 1936-ból származó, elméleti síkja, melyen a gép műveleteket tud végrehajtani, cellánként haladva, egy elméletileg végtelen szalagon haladva. A képzeletbeli gép képes olvasni azt, ami a szalagra van írva, amit értelmez. Miután ez megtörtént, feldolgozza a kapott értéket, és a szabályok alapján mozgatja a cellát a szalagon jobbra, vagy balra, majd megismétli az egész beolvasási folyamatot. Turing így megmutatta, hogy egy egyszerű automatizált folyamat hogyan képes véghezvinni egy eljárást, melynek eredményeképpen egy megbízható számítást kapunk. Legalábbis addig, amíg a gép a szabályoknak egy jól definiált halmazát kapja meg.

Turing jegyzetei szintén egy programozási nyelvet írtak le, akárcsak Gödelnél, de ez már olyan nyelv volt, amit már a mai szemmel is programozási nyelvnek nevezhetünk. Ez a két nyelv azonban nem ugyanaz, erősen eltérnek egymástól. A Turing-féle nyelv nem egy magas szintű programozási nyelv, mint a LISP, inkább egy gépi nyelv, egyesekből és nullákból ez az1936-os találmánya egy borzalmas, ember számára kényelmetlen, használhatatlan gépi kód, amit ma már senki sem használ, ha teheti, az átláthatatlansága és a kezdetleges szabályai miatt.

Ennek ellenére Turing elméleti gépeinek működése egyszerű volt, a gépi nyelve primitív, de mégis rugalmas. Az 1936-os nyelv egy olyan gép működését, írhatta le, amelyet egy élő személy volt képes elvégezni és kiszámítani.

Ezután Turing gondolatmenete drámai fordulatot vett. Feltette a kérdést, miszerint mi az a dolog, amit egy ilyen gép számára lehetetlen megcsinálni? Mi az, amire képtelen? Rögtön megtalálta azt a problémát, amely választ adott a kérdésre, ami nem volt más, mint a megállás. Azaz mikor tudja edönteni magáról egy Turing-gép (vagy egy számítógépes program), hogy megtalálta a kívánt megoldást, és álljon le a munkamenettel.

Ha felveszünk egy időkorlátot, könnyen megoldható a gondunk. Például mondjuk azt, hogy a program álljon le egy éven belül. Futtatjuk ekkor egy évig, és vagy megáll, vagy nem. Amire Turing rámutatott, hogy nagy bajba kerülhetünk, ha azt gondoljuk még a futás elindítása előtt, hogy talál a program megoldást, és nem állítunk be ezért időkorlátot.

Összegezzük Turing elméletét. Tegyük fel, hogy tudunk írni olyan számítógépes programot, ami képes ellenőrizni bármely más programról, hogy megállt-e, és nevezzük ezt „befejezettségtesztelőnek”. Elméletben megadunk neki a bemenetre egy kódot, amit elemezve megmondhatja azt, hogy igen, ez a program be fog fejeződni, vagy nem, ez végtelen időkig futni fog, és soha nem fog megállni. Ezek után készítsük egy második programot, ami a befejezettségtesztelőt használja, hogy kiértékeljen néhány kódot. Ha a kód kiértékelése az első programmal megtörtént sikeresen, akkor a második programunk ezt jelezze, amit így folytatva újabb és újabb programokkal a végtelenségig dolgozhatnánk. És itt jön a fondorlat: adjuk meg bemenetnek az új programunknak önmaga másolatát. Mit fog tenni?

Ne feledjük, ezt a programot úgy írtuk, hogy végtelen ciklusba fog kerülni, ha a tesztelés alatt álló program végrehajtódik. De most önmagát fogja ellenőrizni, így ha végbemegy végtelen ciklusba kerül, azaz nem fog végrehajtódni. Újabb gyönyörű példája az ellentmondásoknak. Lássuk be, hogy az ellenkező kimenet sem fogja megoldani a problémát, hiszen ha nem hajtódik végre, a befejeződéstesztelő ezt jelezni fogja, és nem fog a végtelenségig folytatódni, és megint nem fog végrehajtódni. A paradoxon így azt mutatta meg Turingnak, hogy nem lehet készíteni egy egységes befejeződéstesztelőt.

Érdekes következtetést vont le ebből hamar Turing. Ha nincs rá mód, hogy előzetesen egy számítás alapján eldöntsük, megáll-e egy program, akkor arra sincs mód, hogy bármely más módon meg tudjuk ezt jósolni. Ha nem tudunk tehát sem becsülni, sem jósolni, akkor egy újabb paradoxonba kerülünk, mint az igaz-hamis állítások esetén. Ez az önmagával ellentétes kijelentés hasonlít a számos elméletben Gödel által talált ellentmondásra. (Hiszen ő sem tekintett nagyobb bonyolultságú kérdésekre, csupán a 0,1,2,3... számsorozatra, és két

alapműveletre.) Turing kinyilatkoztatása végső soron megállapította, hogy nem lehet létrehozni teljes formális axiómarendszert.

Miután kitört a második világháború, Turing kriptográfiával foglalkozott, Neumann atombomba robbanásokkal kapcsolatos kalkulációkat végzett, és az emberek megfeledkeztek a formális axiómarendszer hiányosságainak vizsgálatáról jó időre.

Véletlenszerűségek a matematikában

Azon matematikusok korosztálya eltűnt a második világháborúval együtt, akik elmélyültek ezekben a mély filozófiai kérdésekben. Ekkor tűnt fel a színen cikkünk írója, Gregory J. Chaitin.

Az 1950-es évek vége felé, mikor még fiatal volt, olvasott egy cikket Gödelről, és erről a beteljesíthetetlen rendszer elméletről a Scientific American című folyóiratban. Gödel eredményei felkeltették a figyelmét, de nem teljesen tudta felfogni, valami hibádzott a dologban. Turing elméleteihez érve sem került megvilágosodás a homályra, és nem volt elégedett az olvasottakkal. Ekkor jött egy tréfás ötlete a véletlenszerűségről.

Mikor még gyerek volt, Gregory olvasott egy cikket nem egy matematikával, hanem egy fizikával kapcsolatos felfedezésről, a relativitás elméletről és a kozmológiáról, és még többet a kvantummechanikáról. Megtanulta, hogy amikor valami nagyon kicsi méreteket ölt, akkor a fizikai világban nem mindennapi módon viselkedik. Valójában, minden dolog viselkedése előre nem kiszámolható, elég nagy mértékben. Minderről olvasván felötlött a kérdés Gregoryban, hogy vajon léteznek-e hasonló furcsa jelenségek a matematika területén is. Arra gyanakodott, hogy a hiányos rendszer titkának oka ebből eredhet.

Vegyük példának a számelméletet, ahol léteznek elég bonyolult kérdések. Gondoljunk csak a prímszámokra. Minden egyes prímszám máshogy viselkedik, nem kiszámíthatóan, ha jobban elmélyülünk a részletes struktúrájukkal. Való igaz, léteznek statisztikai minták. A prímszámok főtételének nevezik, amely közel pontosan a teljes átlageloszlását a prímeknek, és viszonylag véletlenszerűen viselkedik.

Ezért Gregory tovább vitte a gondolatát, a matematikán belüli véletlenszerűségei jobb megértést szolgálhatnak a hiányosságra. A 60-as évek közepén Chaitin, és teljesen függetlenül tőle a szovjet A. N. Kolmogorov új ötletekkel álltak elő, melyet algoritmikus információs elméletnek szoktunk nevezni. Ez a hangzatos nevű elmélet nagyon egyszerű dolgot tételez fel, miszerint van egy mód arra, hogyan lehet a számítógépi komplexitást mérni.

Az első ehhez hasonló tézis Neumann Jánostól volt hallható. Turing a számítógépet

egy matematikai közelítésű gépnek fogta fel, egy olyan tökéletes gépnek, ami soha nem hibázik, és minden idő és hely szabadon áll rendelkezésre számára. Miután ezt Turing felvetette, a következő logikus lépés egy matematikus számára az volt, hogy vizsgálja meg, pontosan mennyi időre van szüksége a számoláshoz (egy komplexitási számítással). 1950 környékén Neumann rámutatott az időbonyolultsági érték kiszámításának fontosságára, ami mára már egy kellően feltérképezett területté vált.

Chaitin ötlete, hogy nem az időt vette alapul, de nem is vetette el, mivel a gyakorlati időszemléleti megközelítés sem elítélendő. Az ötlet alapja, hogy a számítógépes program méretét figyeljük, az információ mennyiségét amit a gép kap abból a célból, hogy értelmezze a feladatot. Hogy ez miért fontos? Azért, mert a programméret komplexitás szoros kapcsolatban áll a fizika entrópiájával, ami a fizikai rendszerek rendezetlenségi fokával foglalkozik (gondoljunk például a hőtanra). Az entrópia méri a rendellenesség, a káosz mértékét, a véletlenszerűségeket a fizikai rendszerekben. Például egy kristály alacsony entrópiával rendelkezik, ezzel szemben egy gáznak (tegyük fel, szobahőmérsékleten) nagy entrópiája van.

Az entrópiához köthető egy alapvető filozófiai kérdés, hogy az idő miért csak egy irányba futhat? A mindennapokban természetesen nagy különbségek vannak az időben előre és visszafele haladásában. Az üveg eltörik, de nem rakódik össze magától spontán. Hasonlóan a Boltzmann elmélethez, az entrópia nő, azaz a rendszer rendezetlensége fokozatosan növekszik. Ez az elmélet széles körben ismeretes, ez a hőtan második törvénye.

Boltzmann korabeli tudóstársai nem tudták, hogyan lehetne Newton fizikájának eredményeit megfejteni. A gáz atomjainak mozgása, amik úgy pattognak, mint sok-sok biliárdgolyó, észrevehetjük, ezek a mozgások visszafordíthatóak. Ha valahogyan képesek lennénk filmre venni egy kis mennyiségű gáz atomjainak mozgását egy adott időn belül, nem lennénk képesek megmondani, hogy ezt a filmet előrefelé, vagy visszafelé nézzük. Ám Boltzmann gázelmélete kimondja, hogy az idő egy adott irányba mozog mindig, azaz a rendszer egy előre definiált állapotból indul, és egy erősen felkavart rendezetlen állapotba kerül. Erre létezik is egy angol kifejezés, a „heat death”.

Gregory Chaitin és Boltzmann elmélete egymáshoz szorosan összetartozik a számítógépes programok méret és a fizikai rendszer rendszertelenségeinek kapcsolatának köszönhetően. Ebben az esetben egy gázt nagy programmal tudunk leírni, az atomok helyének meghatározása miatt, ezzel szemben egy kristályt könnyen meg tudnánk feleltetni egy kisebb méretű kóddal hiszen a részecskék helye kötött és rendszeres.

Ugyanezen kapcsolat, amely az informatika és a fizika között létezik, fennáll a filozófiai és a tudományos elméletek között is. Ray Solomonoff (egy informatikai tudós, aki még ekkor a Zator vállalatnál dolgozott, Massachusetts-ben) előadta ezt az ötletet az 1960-as konferencián, habár Chaitin csak azután ismerhette meg ezt a szemléletet, miután hasonló

meglátásokra jutott ő is pár évvel korábban. A legegyszerűbb ötle a legjobb ötlet. Nos, mi ez az ötlet? Egy program, ami meg tudja jósolni a jövőbeli észrevételeket, megjegyzéseket. És a legjobb ötlet az, ami felállítja a legegyszerűbb elméletet, amit a legkönnyebben fordíthatunk le egy programba.

És mi a helyzet akkor, ha nem létezik ilyen tömör teória? Mi van akkor, ha a legtömörebb program, ami méretében ugyanakkora, mint az általa feldolgozandó adathalmaz? Ekkor összedől az elméletünk, és az adat érthetetlenné és nem kiszámíthatóvá válik. Az elmélet létrejöttének legnagyobb (és egyetlen) előnye, hogy az adatot, amivel dolgozni akarunk, mindenképpen tömörítsük össze az elméleti következtetések és méretcsökkentő szabályok egy kisebb méretű halmazával.

Tehát definiálni tudjuk már valami olyan dolognak a véletlenszerűségét, amit nem tudunk kisebb méretre tömöríteni. Az egyetlen módja annak, hogy egy teljesen váratlan dolgot vagy számot le tudjunk írni, szükség van egy második emberre, aki ezt megállapítja. Mivel nem létezik semmilyen minta vagy sablonos szerkezet erre, ezért más lényegre törő leírásra nem tudunk támaszkodni. A másik véglet az, amikor van valamink, ami lehet szám is, aminek a mintája gyakran előfordul, rendszeresen. Ezt könnyen felfedezhetjük, például egy szám egymilliomodik ismétlődésekor. Ezek általában hatalmas elemek, egy nagyon rövid leírással.

Gregory Shaitin szerint a kiszámíthatatlan jelenségek felismerésére érdemes a programméret alapján nézni a komplexitást. Mikor megnézzük a program méretét (a program bonyolultságát szem előtt tartva, nem pedig hogy vajon meddig fog futni), egy érdekes dolgot figyelhetünk meg. Bárhol nézzük, hiányosságot látunk. Miért? Azért, mert az elméletünkben a legelső kérdés, amit feltettünk, rögtön ellentmondásba ütközött. Ugyanis kimondtuk, hogy a komplexitást a program méretének nagyságáról állapítjuk meg, amiből mindig megtaláljuk a legkisebb méretűeket. De honnan tudjuk biztosan megmondani, mekkora lehet legfeljebb a legrövidebb számítógépes program? A válasz a kérdésre az, hogy nem tudjuk megmondani. Ez a feltevés szintén két vállra fekteti a matematika következtetéseire épülő erejét.

Hogy megnézzük, konkrétan egy olyan példát, ami ezt az elméletet mutatja be, idézzünk fel egy viszonylag aktuális eredményt. Ha van n darab axiómánk, akkor egy másik, n-nél több darab axiómával rendelkező programról sohasem fogjuk tudjuk bebizonyítani, hogy az a legrövidebb program. Azaz gondban leszünk, ha maga a program nagyobb, mint az axiómák számítógépre felvitt változatának összessége. Még pontosabban szólva, ha nagyobb, mint a program legkisebbségét bizonyító axiómák és méretcsökkentő szabályai.

Könnyen belátható tehát, hogy általánosságban nem érdemes a program méretének bonyolultságát tekinteni elsősorban, mert nehéz meghatározni, és sohasem lehet hajtható végre az elmélet gyakorlatban, ha a program maga nagyobb, mint az axiómáinak halmaza. Ha tudjuk, hogy n darab axiómánk van a legkisebb programnál, akkor semmit sem tudunk az n-

nél több axiómával rendelkező programjainkról, amiből pedig rengeteg van.

Merre haladjunk tovább?

A kinyilatkoztatott következtetések jelentőségük nagy. Három lépésben végigmentünk Gödel elméletén, miszerint kell egy korlátot bevezetnünk, Turingén, aki már konkretizálni is tudta ezt a korlátot időlimit formájában, végül Chaitin elméletében beláttuk, ez a korlát nem lehet a programok méretében sem, és beláttuk a matematika terén a hiányosságokat is.

Nos, ez mind szép és jó. Az algoritmikus információ elmélete egy nagyszerű elmélet, de mondjunk olyan speciális példát, amire nem vonatkozhat a matematikai következtetés ereje. Évekig Shaitin erre a kérdésre azt felelte, hogy talán Fermat utolsó tétele a legjobb példa erre. De egy mulatságos dolog következett be 1993-ban, amikor is Andrew Wiles egy nem mindennapi bizonyítással állt elő. Volt egy apróbb hiba, de mára mindenki belátta, hogy a bizonyítás helyes volt. Szóval tegyük fel, van egy problémánk. Az algoritmikus információs teóriánk megmutatta, sok dolog van, amit nem tudunk bizonyítani, de mégsem tudjuk ezt levezetni független matematikai kérdések megválaszolására.

Hogy lehet az, hogy a hiányosságokon feldühödve, a matematikusok nagy része mégsem próbálkozik megoldani ezeket? A hiányossággal kapcsolatos kutatások elég pesszimista eredményeket szülnek, amivel nem szívesen szembesül senki, hiszen ilyenkor a matematika eszközei sem nyújtanak segítséget. Talán majd a következő generációból egy ifjú tudós be tudja ezt is bizonyítani.

2. A Solow-paradoxon

Az eredményes hardver

A világ egyik legismertebb vitatárgya a Solow-paradoxon. A paradoxon nevét egy Nobel-díjas közgazdászról kapta, aki a következőt állította: *„A számítógépes világ manapság mindenhol megtalálható, kivéve a termelékenységi statisztikákban.”*

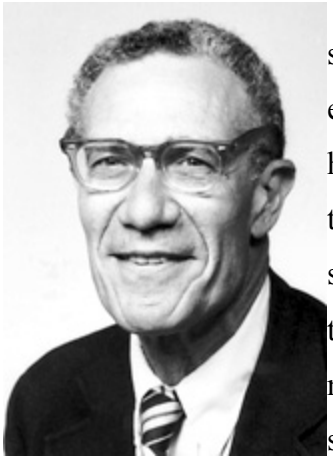
A nagy tiszteletnek örvendő „The Economist” közgazdasági magazin az 1999. július 24.-i számában a híres Robert Gordon professzor e mondatait idézte:

„Az Egyesült Államok közgazdaságának a gyártási részleg termelékenységének a teljesítménye 1995 óta inkább zuhant, mint emelkedett volna. Igaz, hogy a termelékenységi növekedés az 1995 és 99 közötti rövidtávú gyártásban csökkent az 1972-95-ös időszakhoz képest, de ez az emelkedés a hosszútávú gyártásban még jobban lelassult."

Mi az igazság? A vele kapcsolatos túlzott felhajtás, vagy a lesújtó statisztikák? A válasz a kérdésre döntő fontosságú a folyton alakuló közgazdaságra nézve. Ha az informatikai beruházás gátolja a növekedést, akkor mellőzni kell legalább annyi ideg, amíg egy jól működő piac el nem tud bánni a nem várt negatív hatásokkal.

Az eredeti elképzelés szerint az informatikai növekedésgátlás ellen-intuitív. Belátható ekkor legalább az, hogy a számítógépek segítségével sok, egymáshoz nagyon hasonló feladatot gyorsabban oldhatunk meg. A számítógépek által hatékonyabban végezhető a gépelés, rendelés, raktárkezelés, termelési folyamatok felügyelete és a számolás is. Tehát a befektetett erőforrással el kéne érnünk a termelékenységünk javulását. Szóval egyszerűen azt mondhatjuk, hogy ugyanezen emberek többet tudnak majd dolgozni, gyorsabban és olcsóbban elérhetik azt, amit gépek nélkül. Azonban a valóság eltér ettől.

Két fontos tényezőt szoktak elhanyagolni, amikor az informatika előnyeit vizsgálják. Az első tényező az, hogy az informatika fogalma két, egymástól nagyon eltérő közgazdasági tevékenységet is magába foglal. A sok célra felhasználható gép (a PC), és az általa támogatott alkalmazások és egy médium (maga az internet). A főbb pozitívumokra, minthogy különböznek a médiánál megszokottaktól, más gazdasági elvek hatnak. Ezeket külön kell kezelni, és más gondolkodásmódot is igényelnek.



A számítógépes hardver gyártásában masszív, akár két számjegyes mértékű termelési mutató is megvalósítható. Ez elkerülhetetlenül exponenciális robbanás lesz a gépek és a hálózatok erőforrásainak terén. Az informatikát meghatározó két törvény (Moore törvénye, miszerint a chip kapacitása és a számítógép teljesítménye 18 havonta duplázódik; illetve Metcalf törvénye, ami kimondja, hogy a hálózat teljesítőképességének növekedése exponenciálisan nő a csatlakoztatott számítógépek számának megfelelően) szintén lélegzetelállító sebességet diktál a hardverre megjelent informatikai szoftverek esetében. Ezt a jelenséget Robert Gordon időben kimutatta a „Vajon az Új Gazdaság véget vet a termelékenységi lelassulásnak?" című cikkében

(átdolgozva 1999. június 14-ben).

Azonban, hogy a gazdaság kellően felkészüljön a termelékenység megnövelésére, néhány feltételnek teljesülnie kell. A régiből új technológiába vezető átmenetnek (a számítógép elég gyorsan válik elavulttá) nem szabad túl „kreatívan rombolónak" lennie. Ugyanis ezek a változások óriási méreteket öltenének, mint például a régi hardvertől való megszabadulás költségei, a kezelési technikák módosítása és új technikák bevezetése, az új alkalmazottak betanítása a képzetlenek és a képezhetetlenek helyére, az új hardver és szoftver telepítése és a vállalat minden szintjén a dolgozók képzése. Hosszú távon nem veszélyeztethetik ezek a kiadások az új technológiából várható bevételeket. Azaz felmerülhet kérdések, hogy vajon többe kerül bevezetni, működtetni és fenntartani azokat az informatikai eszközöket, amiket biztosan le akarunk cserélni? Vajon meddig tart a felzárkózás az új technológiához a jelenlegi informatikai ismereteinket tekintve, elég időnk lesz megszokni a különbségeket a régi és új között? A technológia képes lesz-e kinőni a gyermekbetegségeit (többek között a megbízhatatlan üzemeltetés, rossz tervezés, pontatlanság, az első generációs számítógép felhasználók hozzánemértése és a felhasználóbarátság hiánya)?

Továbbá, a bevezetendő informatikai technológia csupán egy újabb, kis lépés a fejlődés irányába, vagy egy teljes újjászületésként jelenik majd meg? Tevékenységekei által hasonló dolgokat tudunk megvalósítani eddig eltérő módon, vagy eddig még nem ismert távlatok nyílnak meg az emberi képzelőerő és kreativitás irányában? A válaszok és az előrejelzések eléggé változatosak.

Az informatika tudománya nem volt olyan széles mértékű az emberiség számára, mint az elektromosság, a belső égésű motor vagy a telegram feltalálása. Az sem teljesen nyilvánvaló, hogy egy univerzális jelenségként létezik-e, ami mindenféle országhoz és mentalitáshoz egyformán illeszkedik-e? Az informatika és a médium áttörése nem volt egyforma sehol a világon, annak ellenére, hogy a felhasználható erőforrások hasonlóak voltak majdnem miden szintű vállalati osztályon. A volt kommunista országoknak is ezt kellett volna figyelembe venniük. A gazdaságuk túl elavult és túl maradi, szegény és rosszul kezelt volt, hogy egy ilyen kritikus változást vigyen végbe az IT ügyében. Egy rossz piacra vagy vállalatra dobott technológia gyakran gátolja a haladást, vagy válik terméketlenné.

Az internet története és paradoxona

A másik tényező pedig nem más, mint az internet. Az internet számítógépeken üzemel a gépek számára oly módon, ahogyan a TV-műsor a tévékészülékek számára. A hasonló a

kettőben pedig az, hogy elfedi a valóságot, és nagyon gyakran félrevezető. Példának okáért a szolgáltatási szektor termelékenységét lehetetlen mérni, nem is szólva arról, hogy az internet határtalanul informatív és dinamikus is. Világossá vált mindenki számára, hogy az internet egyfajta médium, és mint olyan, része az elődjeivel együtt a nagy átalakulási folyamatoknak. Közép- és Kelet-Európa nem olyan régóta kapcsolódott ehhez a folyamathoz, szemben a folyton előttünk járó Egyesült Államokkal.

Az internet nem más, mint a legmodernebb hálózat, ami forradalmasította az életünket, akárcsak az egy évtizeddel előtt beharangozott távíró és a telefon, amik szintén globális változásokat idézett elő. Általános értelemben mire számíthatnak akkor a közép-kelet-európai országok az internettől világszinten és határaikon belül? A probléma jelen helyzetben nem a termelékenységi tényezőktől fog függeni, hanem továbbra is az üzleti folyamatoktól.

Ahogyan említettük, minden kommunikációs médium hasonló fejlődési cikluson megy keresztül.
Ezen ciklus lépése pedig a következők:

Első lépés, az anarchia, avagy a publikus fázis

Ebben az lépésben a médium és a hozzá kapcsolódó erőforrások olcsóak, elérhetőek, és nincsenek szabályok által megszorítva. Megjelennek a publikus szektor szereplői, mint például a felsőfokú oktatási és vallási intézmények, önkormányzatok, a nonprofit és civil szervezetek, szakszervezetek. Ekkor még a korlátolt pénzügyi erőforrások miatt túl költséges arra a célra, hogy mondanivalójukat szélesebb körben terjesszék.

Az internetnek is át kellett esnie ezen a kezdeti fázison, ami számára egy folytonos haláltusa volt. Az elején egy testet öltött számítógépes zűrzavarként létezett, ami ad hoc hálózatokból, helyi és szervezeti hálózatokból épült fel (elsősorban az egyetemek, a kormány által létrehozott szerv, a DARPA és az USA biztonságért felelős központjainak a hálózatai). Néhány nem kereskedelmi szereplő megragadta az alkalmat és (a kormány által támogatott adományokkal) el kezdte összefűzni ezeket a kisebb hálózatokat. Az eredmény így egy világszinten kialakított egyetemi hálózat lett. Az amerikai Pentagon kiépítette az ARPANET-et, az összes hálózat hálózatát. A kormány egyéb részlegei is ringbe szálltak, élükön a Nemzeti Tudományos Alapítvánnyal (röviden NSF), ami igencsak későn vonult vissza az internettől. Az internet így köztulajdonná vált, amihez egy igen csak szűk réteg korlátlanul hozzáférhetett.

A rádióközvetítés is hasonló folyamaton ment keresztül. 1920-ban kezdődtek meg az első adások az Egyesült Államokban. Ezek áttekinthetetlenek, zűrzavarosak és

rendszertelenek voltak. A nonprofit szervezetek saját adásokat kezdtek sugározni s kis költségvetésű, helyi rádióadókat építettek ki. Szakszervezetek, néhány oktatási intézmény és vallási csoportok hoztak létre nyilvános közvetítéseket.

Ezt követte a kereskedelmi fázis

Mikor a rádióhallgatók és a számítógép felhasználók tömege erősen megnövekedett, az üzleti szektor elérkezettnek látta az időt, hogy csatlakozzon. A kapitalista ideológia nevében megkövetelte a médium privatizációját. Ez nagyon érzékeny pontokat érintett meg minden nyugati ember lelkében. Ezek a pontok az erőforrások egyenlő kiosztása a versengés számára, a publikus szektorhoz kapcsolódó korrupció és elégtelenség („Other People's Money” - OPM), a vezető politikai rétegek tagjainak hátsó, ártó gondolatai (a híres amerikai paranoia), némely közösség sokrétű ízlése kielégítésének hiánya, a demokrácia és a magánvállalkozás egyezőségei.

A végeredmény hasonló. A magánszféra vagy meghódítja médiumot alulról (visszautasíthatatlan ajánlatokat tesznek a médiumot irányítóinak vagy tulajdonosainak), vagy felülről (megfelelő befolyással a hatalom kiskapuiban elérik, hogy a médiumot törvények által privatizálják).

Minden üzleti kisajátítás, különösen a médiáé nyilvános elégedetlenséget váltott ki. Számtalan valós és valósnak vélt gyanú támadt azzal kapcsolatban, hogy a közösség javait az elüzletiesedés el fogja tiporni. A monopolizációtól és a kartellizációtól való félelemben egyre inkább jogosabbnak tűntek ezek a gyanúsítgatások. A médiumot veszélyeztette továbbá az is, hogy az eddigi közösségi irányítást a kiválasztott emberek maroknyi csapata veszi át. Ez sajnos mind be is következett kivétel nélkül, ám a folyamat oly lassan ment végbe, hogy a kezdeti félelmeket elfeledték és az emberek figyelme átterelődött az aktuálisabb problémákra.

1934-ben egy új kommunikációs törvényt vezettek be az Államokban. Ez kimondta, hogy a rádiófrekvenciákat nyilvánítsák nemzeti erőforrásnak, és adják el a magánszférának, amit mostantól csak a rádiójeleket vevő, illetve adó, csak fizető hallgatóknak és adóknak lehessen továbbítani. Másképp szólva a rádiót az üzleti világ átadta magánkézbe, amivel vége is lett a publikus rádióadásnak.

1995 áprilisában az amerikai kormány visszavonta az utolsó nagyobb beruházását az internet fejlesztésétől, amikor az NSF nem finanszírozta tovább bizonyos hálózatait. Sőt, a kormány privatizálta az eddigi jelentős nettel kapcsolatos befektetéseit is. 1996-ban egy újabb kommunikációs törvényt hoztak, mely engedélyezte a „rendszerezett zűrzavart”. Azaz

lehetővé tette a média vezetőinek, hogy betolakodhassanak mások területeire. Például a telefontársaságok továbbíthattak videót és a kábeltársaságok közvetíthettek telefonálásokat. Hosszú időn át ezt fázisokra osztották, mégis ez egy olyan gyökeres változás volt, amelynek a jelentőségét nehéz felmérni és amelynek a következményei minden képzeletet felülmúltak. Ez magában hordozta a hivatalos cenzúra szükségességét is. Meg kell hagyni, hogy a szándékos cenzúrázás valahogy "fogatlan szabványosítás" és "végrehajtó hatalom" szerepét játszotta, mindazonáltal meg kell hagyni, hogy olyan cenzúra volt, amely keményen fellép saját intézményei ellen. A magánszféra perekkel fenyegetőzött, de a felszín alatt a nyomás és a kísértés hatására elkészítette a saját cenzúráját a kábeltévés és az internetes médiában.

A harmadik fázis az intézményessé válás

Ennek a fázisnak a jellemzője a felerősödött törvényhozási események sorozata. A törvényhozók mindenféle szinten el kezdik felderíteni és változtatni a médiumot. A korábban szabad erőforrások hirtelen „nemzeti kincsekké” alakulnak át, melyeket nem lehet csak úgy félvállról, könnyelműen bárki k számára szétosztogatni.

Ezek alapján elképzelhető, hogy az internet bizonyos részei államosítva lesznek (például szerződési követelmények formájában), és fel lesznek ajánlva a magánszektornak. Törvénybe lesz szintén iktatva, a net szereplői számára mely tartalmak lesznek engedélyezettek és nem megengedett, tiltottak (mi számít trágárságnak, felbujtásnak, faji és nemi előítéleteknek).

Az USA-ban (nem is beszélve a világ más részeiről) egy médium sem kerülte el a törvényesítést. Mindig lesz mód rá, hogy a különböző idő-, hely-, szoftver-, tartalom-, hardver- vagy sávszélesség igényeket el tudják osztani a nyilvánosság, az üzleti közösségek és a fent maradó, kisebb rétegek között. Ez az ára annak, hogy az üzleti szektor elhárítsa a túlbuzgó törvényhozók kellemetlenkedéseiket. Ezek összessége vezet a hosztok és szerverek monopolizációjához. A lényeges, fontos adók száma csökkenni fog, amelyek tartalmát szigorú megszorítások fogják cenzúrázni. Azokat az oldalakat, amelyek nem tartják be ezeket, törölni vagy közömbösíteni fogják. A tartalmi útmutatások (a cenzúra szebbik megnevezése) már régóta léteznek (CompuServe, AOL, Prodigy).

Az utolsó, végső fázis pedig a „Vérfürdő”

Ez a fázis a megszilárdulási fázis. A játékosok száma erősen megfogyatkozik. A böngészők típusainak száma lecsökken kettő, vagy esetleg három fajtára (Mozilla Firefox, Intenet Explorer és talán egy harmadikra, Chrome vagy Opera). Egyre több hálózat összeolvad egy nagy, magánkézen működő óriáshálózattá, a szerverek pedig szuperszámítógépeken futó hiperszerverekké. Az internet-hozzáférés szolgáltatók (ISP) erősen megfogyatkoznak. 1983-ban az USA médiapiacának nagy részét 50 vállalat uralja. 1995-re ez a szám 18 lesz, a század végére pedig még kevesebb.

Ekkorra a pénzügyi életben fennmaradásért harcoló vállalatok a felhasználók, hallgatók és nézők megszerzéséért versenyeznek. A programozás lealacsonyodik a legalsóbb (és legelterjedtebb) közös nevezővé, és válik uralkodóvá, amíg a vérfürdő tart.

Utólag visszatekintve megérthetjük talán, miképp tökéletesítette a számítógép azon képességünket, hogy egy dolgot többféleképpen és sokkal eredményesebben csinálhassunk meg. De egyvalami gyorsan tisztázódott. Az informatika adta előnyök függhetnek a történelmi, szociális és gazdasági viszonyoktól, függetlenül a technológiától. Igazi paradoxon ez, hiszen egy tisztán az előrehaladással foglalkozó terület könnyen okozhat erős meghátrálást is, ha nem megfelelőek a viszonyok.

Mikor mutatják be, hogyan mutatják be, milyen célokkal használják fel, sőt, még az is lényeges lehet, ki mutatja be – ezek jelentősen meghatározzák a későbbi bevezetés költségeit, és ebből következően erősödik a megvalósíthatósága és a közreműködése a termelékenységben. Mindenképp érdemes ezt a Közép-Kelet országainak, köztük hazánknak is megjegyezni és betartani.

Egy remek aktuális példa a Solow-paradoxonra a 2010 decemberében Putyin orosz elnök által kiadott felső utasítás, amiben elrendelte, hogy az orosz közigazgatási intézmények, hivatalok és közigazgatási egységek hozzanak létre egy működőképes tervet a Linuxra való áttérésre. Ennek a tervnek a lényege, hogy 2012 áprilisától Oroszország költségvetési intézményei a nyílt forrású rendszerekre váltana a kereskedelmi szoftverekről. Ha párhuzamot vonunk itt a Solow-paradoxonnal, akkor láthatjuk az első fázist, melynek során haladás céljából bekövetkezik a változás. Viszont a haladás itt is akadályokba ütközik annyira, hogy az nem előrébb vinné a közintézmények informatikai rendszereinek hatékonyságát, még vissza is fogja azt.

Az ország a Linux alapjain egy saját, a Windows leváltására alkalmas operációs rendszert kívánt kidolgozni. Ez a rendszer olyan erősre volt tervezve, hogy még a Microsoft mamutcégnek is tisztes konkurenciája lehetne a piacon. A Linux rendszer disztribúciói már bizonyítottan jobban remekelnek a Bill Gates által vezetett cég termékével szemben biztonság és megtakarítás terén is (nem is beszélve a redmondi függőség csökkentéséről). Hasonló

terveket adott ki a kormány sok más európai országban is, de az évek során kivétel nélkül elbuktak. Mi ennek az oka? A korábban nyílt forrású rendszerekkel kísérletező Németország, Ausztria, Franciaország és Nagy-Britannia ugyanis belátta, hogy üzemeltetése sokkal költségesebb lenne, és bizonyos feladatokra egyáltalán nem alkalmas. Egyik leghangosabb bukást Bécs könyvelhette el 2005 elején, ahol a visszaállítás is plusz kiadásokat jelentett. A migráció 2007-ig összesen ezer gépen sikerült, majd ezek nagy részét vissza is állították. Ehhez a példához kísértetiesen hasonlít a Linuxra átállított londoni tőzsde (LSE) esete is, ahol már az első nap tetemes mennyiségű hibával indított a rendszer.

"*A Millenian Exchange működésének hétfői megkezdését követő 20 másodpercen belül már riasztást is küldtek rendszereink a kereskedők adatait közötti jelentős eltérések miatt. Az eltérés jóval nagyobb volt, mint amit valaha is láttam, és még sokkal nagyobb, mint amit ugyanazok a kereskedők más tőzsdéken tapasztaltak. Riasztó volt*", közölte egy a kereskedést folyamatosan felügyelő ún. benchmarking cég munkatársa. (Forrás: Pcforum.hu)

Mindezen hibák ellenére a Linuxot, mint legnépszerűbb nyílt forrású rendszert nem ásták el az országok kormányai, inkább félretették későbbi időre, amikor már a kiforrottabb pingvines szerverek jobban megbirkóznak a rájuk rótt feladatokkal.

3.A Homonkulusz-paradoxon és a vezérlő egység paradoxon

Definíció

Nem gyakran találkozunk a homonkulusz szóval a hétköznapi életben. Bár sok mindenkinek ismerősen csenghet a szó, a jelentésével nem mindenki van teljes mértékben tisztában. A latin eredetű szó jelentése: kicsi ember, emberke, méghozzá eredetileg lombikban, vegyi úton előállítható emberszerű lény. A definíciót főként az egykori alkimisták, mint a híres Paracelsus hozta létre, ám később a biológiában kapott nagy szerepet. Ez az emberi születéshez kapcsolódik, miszerint minden spermiumban egy kis homonkulusz rejtőzik, és emberré válása során nem folyamatosan fejlődik, hanem csak részeiben és egészében növekszik.

A mai tudományos területeken az emberi test egy olyan skálázott modellje, ami valamilyen módon illusztrálja a fizikai és pszichológiai , és egyéb absztrakt és elvont tulajdonságait, jellemzőit és működését az emberi testnek. Jelen példánkban egy rendszer

működését jelenti, míg a hozzá kapcsolódó paradoxon kicsit csavar a definíció, továbbgondolja a jelentését:

„A homunkulusz-paradoxon egy filozófiai ellentmondásféleség, amely számos tudomány (a biológia, a pszichológia, a logika, a kognitív tudomány, az filozófiai ismeretelmélet, a számelmélet) területén megjelenik, és minden esetben kísértetiesen azonos érvelésforma vezet hozzá, olyan úgynevezett regressus ad infinitum érvelés, amely a végtelenségig folytatható. Gyakran használják egy elmélet vagy egy elmélet vitatott mondatának ellentmondásosságának megvilágítására; akkor alkalmazható, ha az illető vitatott mondatból következik, hogy egy, az elmélet által vizsgált objektum az elmélet által meghatározott értelemben "tartalmazza önmagát".

Ha így fogjuk fel a dolgot, akkor tulajdonképp az is mondható, hogy homunkulusz paradoxon egyszerűen egy regressus ad infinitum-érvelés. De a „tartalmazás” fogalmának fent említett önkényes kiterjesztése vitatható.”

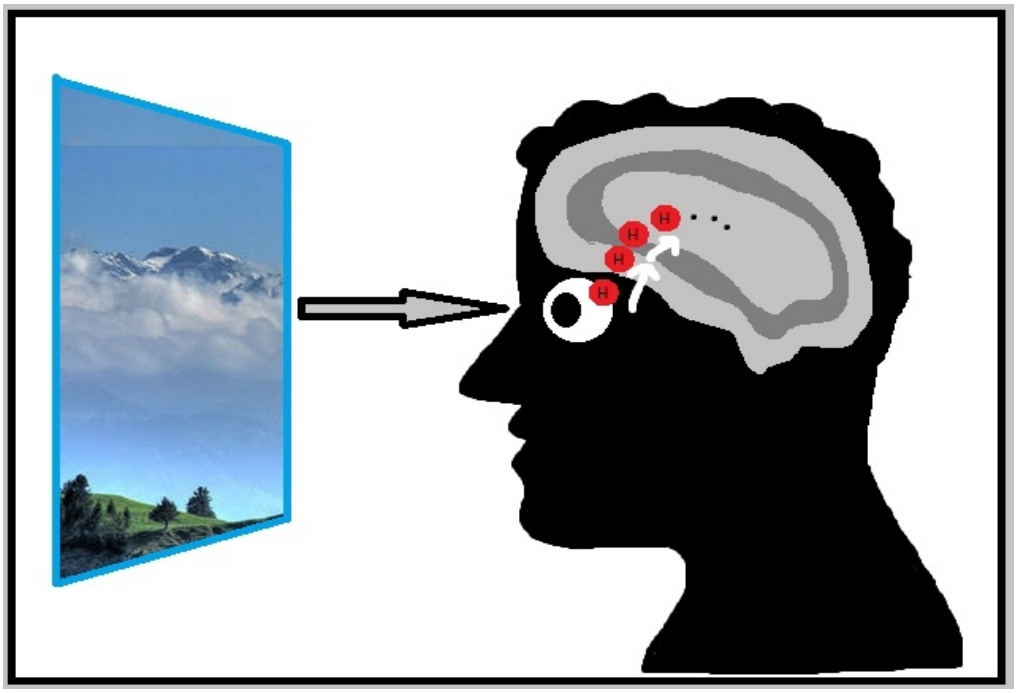
Mint azt a definícióban is olvashatjuk, a homonkulusz-paradoxon fogalma már nem a biológia területén bír jelentéssel, de magát a gondolatot, miszerint valamilyen rendszer már magában tartalmazza egy rendszer tulajdonságait, továbbviszi az elméleti és a gyakorlati tudományok irányába. Előbb ismerjük meg a filozófiai, és a humán jellegű megközelítést!

<u>A homonkulusz vita</u>

A homonkulusz vita Richard Gregory-tól származik, 1987-ből, amely kitűnően szemlélteti a jelenséget definiáló jelenség értelmezését. Ezek a viták mindig megtévesztőek. Gyakorlati hasznuk a pszichológia és a filozófia terén abban rejlenek, hogy kimutatják, mely elméletek vannak bukásra ítélve, vagy mely elméletek inkompetensek.

Ezek a viták gyakran előfordulnak a látomás, a víziós elméletekben. Képzeljünk el egy embert, aki éppen egy filmet néz. Képeket lát, amiket gondolatban elkülönít az elméjétől, az általa érzékelt külső világ részeként fogja fel, amik a vászonra vannak kivetítve. Hogyan történik ez? A többség által elfogadott egyszerű elmélet szerint a kivetítőről visszavetődő fény alakot öltenek az emberi szem retinájában, és ezeket valami az agyban úgy tekinti, mint a vászonra vetített filmet. A homonkulusz vita azonban arra mutat rá, hogy ez nem lehet a teljes és hiánytalan magyarázat, mert ezen folyamat egy, a látó szem retinája mögött létező személy,

vagy homonkulusz cselekedeteinek köszönhetően megy végbe. Egy mesterkéltebb vita pedig állítja, hogy a retinában megjelenő képek egy vizuális dolgok megjelenítésért felelős kéregállományba jutnak, ahol beolvassa őket az agyunk. Ám ismét nem lehet ez a teljes magyarázat, hiszen hogy ez mind végbemenjen, egy újabb emberkének kellene lennie a kéregállomány mögött, ami ezt a folyamatot felügyeli. A látáshoz kapcsolódó elméletekben a homonkulusz vita érvényteleníti a nem projekciós nézeteket, melyek szerint a valódi nézőpont nem egyezik meg az általunk látott dolgokkal (szintén Gregory kijelentése, 1990-ből).



Ryle 1949-ben megjelent teóriája a végtelenségig folytatható vitáról hasonló gondolatokat vetett fel. A hiedelemmel összhangban ugyanis, valahányszor egy személy valamilyen cselekedetet intelligensen hajt végre, a cselekedetét egy másik, szabályok alapján működő és ezek alapján javaslatot tevő belső cselekedet irányítja és hagyja jóvá. Ezért ki kell jelentenünk, hogy az illető egyén válasza azon kérdésre, hogy mely cselekedet lesz ésszerű és értelemszerű, egy már előzőleg végrehajtott ésszerű és értelemszerű cselekedet alapján fog végrehajtódni. Ennek a belső regressziónak a végtelensége prezentálja, hogy megfelelő alkalmazása nem vonja maga után eme kritérium működésének létjogosultságát.

Röviden tehát Ryle teóriája szerint amennyiben létezik egy belső elme vagy entitás, akkor ennek biztosan nem lehet része az intelligens cselekedet. John Searle „Kínai szobája” pedig azt a tényt demonstrálja, melynek állítása szerint az intelligencia öntudatlan, vagy az intelligencia megjelenése nem tartalmazza magában az intelligencia létezését. Tehát ha az ember intelligens, akkor a homonkulusznak nem szükséges ellátni a központi szerepet. A

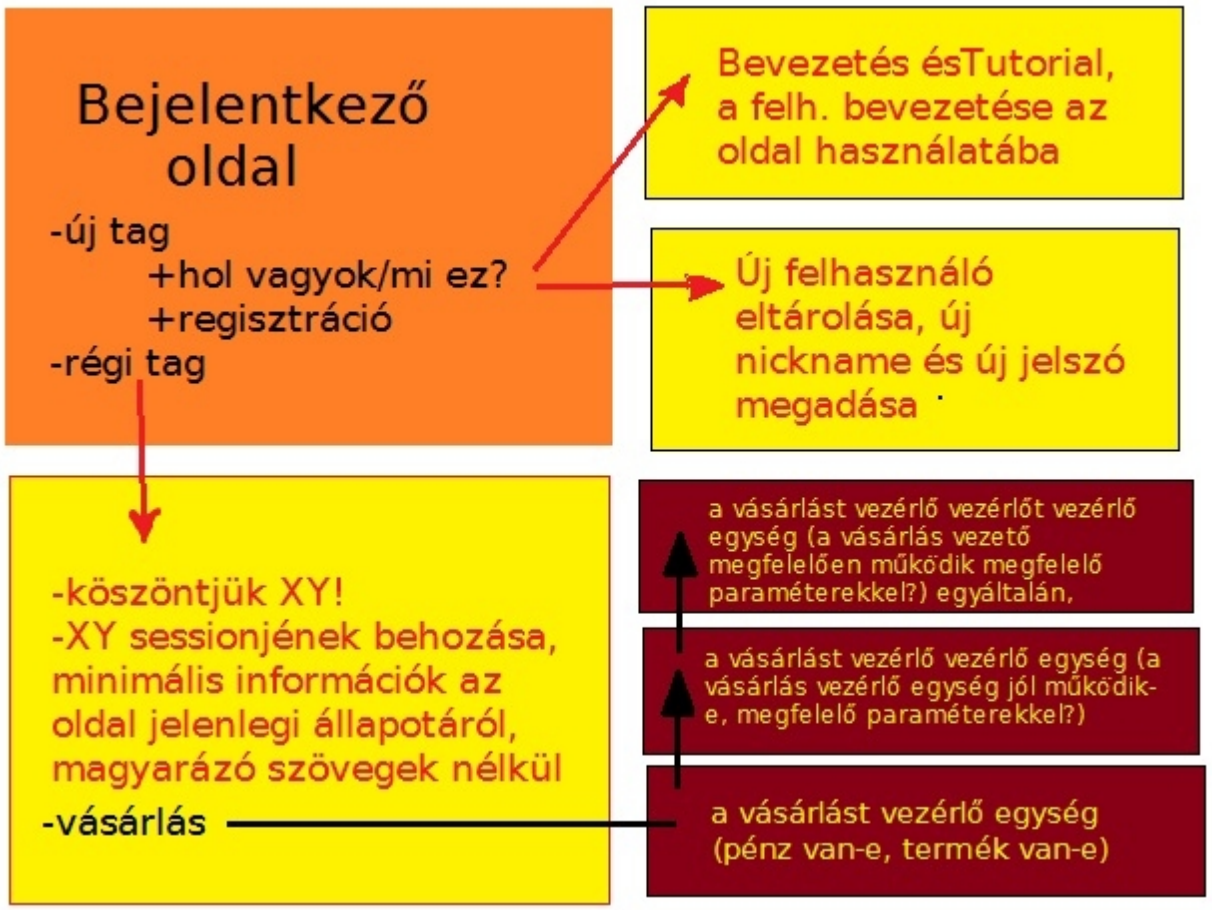
homonkulusz vitát és a regresszív vitát gyakran összetévesztik egymással, de nagy a különbség. A homonkulusz vita azt mondja ki, hogy ha szükség van egy „apró emberkére”, ami bebizonyítja és kiegészíti az elméletet, ha az hamis vagy hiányos. Míg ezzel szemben a regresszív vita azt állítja, hogy az intelligens egyénnek gondolkodnia kel, mielőtt gondolata támadhatna.

Homonkulusz-paradoxon az informatika területén

Az informatika terén nem találkozunk ilyen mély filozófiai kérdésekkel és felvetésekkel. De szintén megjelenik az informatikában is. Ennek konkrét példája Help jellegű fájlok, vagyis a felhasználóknak nyújtott segítség témakörében. Ha egy kezdő felhasználó szeretne segítséget valamilyen szolgáltatásról (például a Wikipédia), el kell olvasnia a szolgáltatáshoz adott segítséget (például Wikipédia: Segítség lap).

De ez a segítség ugyanolyan környezetben, ugyanabban a rendszerben létezik, mint maga a rendszer (a Wikipédia:Segítség lap maga is része a Wikipédiának – de ugyanúgy a DOS rendszer szolgáltatásaihoz nyújtott segítség is egy DOS paranccsal hívható, illetve Linux esetén a terminálból kell indítani a man parancsot). Tehát hogy ezt a segítséget igénybe vegye, igénybe kell vennie egy újabb segítséget, amely leírja, hogyan működik a segítség – egy Segítő Meta-Szolgáltatást. Ám ha ez a Segítő Meta-szolgáltatás (a Wikipédia:Segítség laphoz írt Wikipédia:Segítség:Segítség lap lenne) is része a rendszernek, szükség van egy Segítő Meta-Meta-Szolgáltatásra, és így tovább, a végtelenségig.

A rendszerhez nem értő felhasználó soha nem fog tudni mit kezdeni a rendszerrel, ha a rendszer alkotói nem tudnak a *rendszeren kívüli* segítő szolgáltatást nyújtani. Ilyen rendszeren kívüli szolgáltatás lehet egy felhasználói kézikönyv a szoftverhez, esetleg egy már felhasználó által ismert szoftvert és az új szoftvert összekapcsoló interfész, vagy jelen esetünkben egy külön segítő, úgynevezett „Tutorial” oldal. A bejelentkezés előtt a honlap látogatója eldöntheti, szeretné-e előbb megismerni a szolgáltatásokat, vagy már ismeri őket, és szeretne a lényegi részre térni-e.



A Homonkulusz-paradoxon-rész lesz a „Hol vagyok/ Mi ez?” elágazás, ahol a bevezetésben és a tutorialban az új felhasználót tájékoztatja az oldal funkcióiról. A mesterséges intelligencia vezérlő elve pedig a három sötétebb színnel jelölt php oldal lesz, ahol a végtelenségig is lehet majd fokozni a vásárlást vezérlő egységek egymásra épülő hierarchiáját.

4. Paradoxonok egy különleges WebShopban

Mielőtt egészében vizsgálnánk a honlapot, ismerkedjünk meg a bonyolultabb beépített függvényekkel.

Fő index.php kódhoz szükséges leírások, magyarázatok

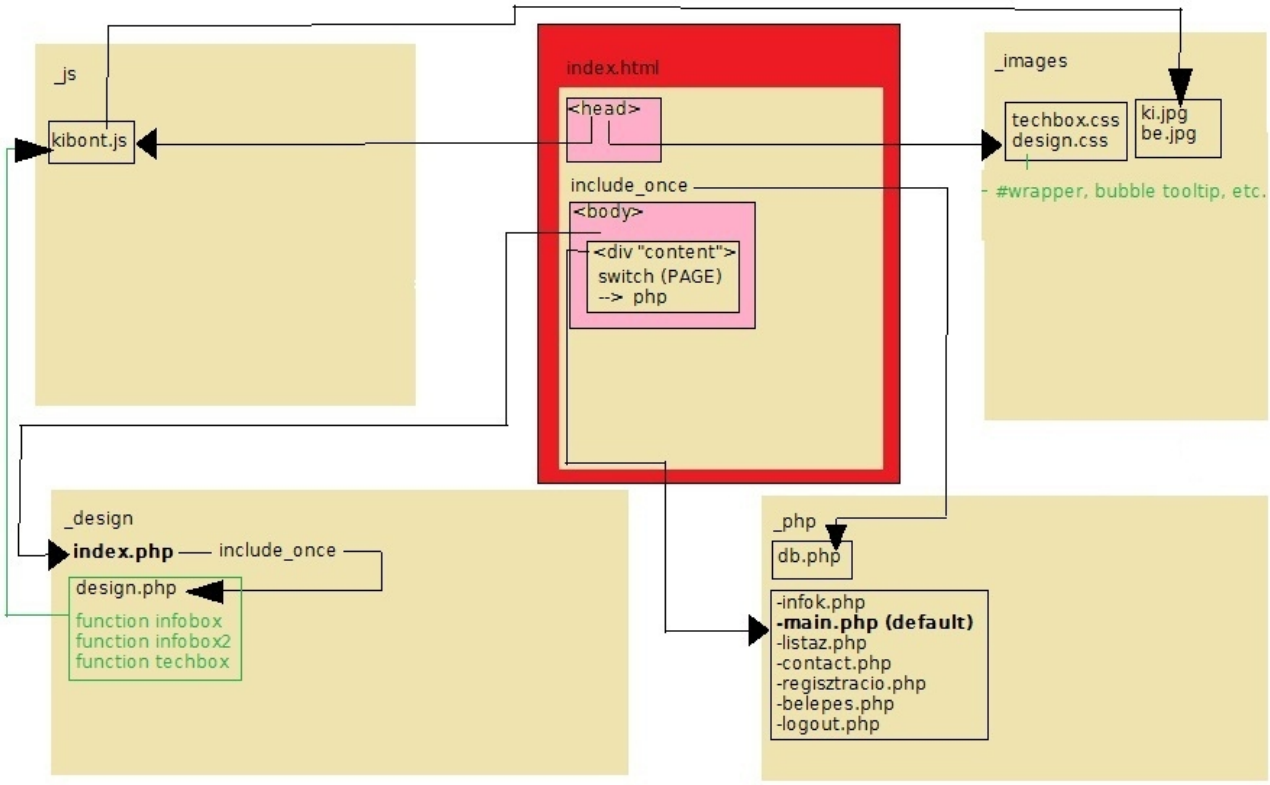
Honlapunk központjaként, szíveként szolgál ez a terjedelmes php kód, ezért fontos, hogy mindenekelőtt ezzel legyünk tisztában.

A void **ob_start** ([string output_callback]) függvénnel kapcsolhatjuk be a PHP saját

belső kimenet pufferelési mechanizmusát. Ameddig a kimenet pufferelés be van kapcsolva, a fejléc adatokon kívül semmiféle kimenet nem hagyja el a PHP háza tájékát, az egy belső pufferben tárolódik. Ha végül szeretnénk a puffer tartalmát kiadni a php kezéből, akkor jön szóba az ob_end_flush().

Az include_once() tartalmaz és kiértékel egy különleges fájlt, a script végrehajtásakor. Viselkedése hasonló az include()-hoz, azzal a különbséggel, hogy ha a fájl kódját már eleve tartalmazza, akkor nem fogja még egyszer bekérni. Ahogy a neve is mondja, „tartalmaz_egyszer”.

A beépített $_GET függvényt arra használjuk, hogy get metódus által kiküldött formból értékeket gyűjtsünk ki/szedjünk ki.



A belepes.php függvényeinek leírása

A belepes.php a kód egyik legérdekesebb része, ahol lehetővé tesszük, hogy minden vevő beléphessen a már regisztracio.php-ban lévő felhasználónevével és jelszavával.

 Belépés<hr />

<?php

/*Ha még nem lépett be**/

if (!$isLoggedIn)

```
{  if (isset($_POST['login']))
  {   $nick = addslashes($_POST['nev']);
```

/*Visszaad egy sztringet, amiben visszaperjelet (\) tesz azon karakterek elé, amiket adatbázis lekérdezésekhez speciálisan kell kezelni (aposztróf, idézőjel, visszaperjel (backslash)és NULL). Egy jó példa az addslashes() használatára, amikor adatot viszünk be egy adatbázisba. Például az O'reilly névnél az idézőjelet kezelni kell.*/

```
   $pass = md5($_POST['jelszo']);
```

/*Kiszámolja az MD5-ös biztonsági kód megfelelőjét az adott sztringnek (jelen esetben a jelszónak).*/

```
   $sql = "SELECT * FROM user ";
   $sql.= "WHERE `username`='".$nick."'";
   $sql.= " AND `passw`='".$pass."';";
   $query = mysql_query($sql);
   if (mysql_num_rows($query) !== 0)
   { /*Helyes nick és a pass, azaz a query eredményt adott vissza.*/
     $_SESSION['nick'] = addslashes($_POST['nev']);
     $_SESSION['belepett'] = true;
     header("Location: ".$_SERVER['PHP_SELF']);    }
```

/*A header() arra való, hogy nyers http header kódot tudjunk vele elküldeni. Ebben az esetben a $_server php-self-jére. */

```
   else
   {/*Hibás nick, vagy pass.*/
     print "hibás nick/pass";    }  }
?>
```

[… Formok, ezek triviálisak...]

```
<?php
} /*Ha már belépett*/
else
```

{/*Ha be van lépve, kiírjuk a nicknevét és az index.php-ban a p értéke logout lesz, ami behozza a logout.php-t a _php mappából.*/

print "bejelentkezve: ".$_SESSION['nick'];

print " kijelentkezés"; } ?>

A contact.php, infok.php, listaz.php és main.php függvényeinek leírása

Simán létrehozunk egy-egy span-t, Contact és Információk vagy main esetében Főoldal néven, majd berakhatunk képeket az _images mappából. A listaz.php-ban a span után egy php-blokkban ismét ellenőrizzük, hogy be vagyunk-e logolva. Ha igen, akkor értesítsük a felhasználót a listázásról, ha nem, akkor szóljunk, hogy a felhasználó nincs bejelentkezve.

A db.php függvényeinek leírása

Itt definiáljuk az adatbázisunkhoz való kapcsolódás paramétereit.

```
<?php
    function start_db() {
    $db_url = "127.0.0.1";
    $db_name = "Makettbolt";  /*Az adatbázisunk neve.*/
    $db_username = "root";   /*Az adatbázisban a főadmin felhaszálóneve.*/
    $db_password = "";  /*A főadmin jelszava, jelen esetünkben nincs.*/

  $rs=mysql_connect( $db_url, $db_username, $db_password );   /*Kapcsolódás*/
  mysql_select_db( $db_name );
  return $rs; }
?>
```

Elindítjuk a start_db() függvényt, ami úgymond beindítja az adatbázist. Az adatbázis URL címét a $db_url változóban adjuk meg, továbbá az adatbázis nevét, felhasználónevet, és jelszót. Ez az adminisztrátor esetében „root” lesz.

Ezután létrehozunk még egy változót, $rs néven. Ez egy adatbázis kapcsolódás lesz, aminek három paramétere az URL-cím, a felhasználónév és a jelszó, és ezt az adatbáziskapcsolódást fogjuk visszaadni rs néven a blokk végén. De előbb mysql_select_db függvénnyel kapcsolódunk a paraméterben megadott adatbázisnévhez ($db_name).

Így a blokk végén rs visszaadásával az index.php-be include-dal meg tudunk adni egy érvényes, mégis dinamikusan változtatható adatbázist.

A logout.php függvényeinek leírása

Egyetlen php blokkra van itt szükségünk, amiben visszavonjuk, pontosabban megsemmisítjük a belépett és nick sessionöket, és visszajuttatjuk az irányítást az index.php-ra.

```
<?php
unset($_SESSION['belepett']);
unset($_SESSION['nick']);
header("location: index.php");
?>
```

A reg_check.php függvény

Igazából a reg_check.php is egy php, de egyetlen egy függvényből áll, ami a regisztrációnál létrejövő komplikációnak megfelelően dob vissza valamilyen üzenetet. Nézzük sorba, mik a lehetséges variációk! A reg_check.php természetesen a regisztráció.php-re épül, az ott létrehozott adatok tömböt kéri be paraméterül.

1. A felhasználónak meg kell adni a regisztráció során nicket (milyen néven szeretne bejelentkezni az oldalra), jelszót, valódi nevet, e-mail címet, lakáscímet és telefonszámot(ezek a rendelés kiszállításakor lényegesek). Ha ezekből BÁRMELYIKET is kihagyta, akkor üzenetben megkérjük, hogy töltse ki mindet, mert valószínűleg egyet vagy többet kifelejtett, és ekkor nem lesz sikeres a regisztráció.
2. A nicknevet megvizsgáljuk. Ezt egy literálsorral ellenőrizhetjük: (^[a-zA-Z\-_áéíóöüóűÁÉÍÓÖÜŐŰ]+$). Ez azt jelenti, hogy ha kis vagy nagy betűvel, vagy

kötőjellel, aláhúzásnévvel, vagy ékezetes betűkkel kezdődött (és utána tetszőlegesen folytatódott), akkor ez egy érvényes nicknév. Ezt úgy is vizsgálhatnánk, hogy számmal kezdődött-e: ^[0123456789]+$, de ekkor még az összes nem érvényes karaktert is fel kéne sorolni, ami nemzetenként változik a billentyűkiosztások miatt, és ezzel lehetetlen feladat elé állítanánk programunkat.

3. E-mail címet hasonlóan a nicknévhez literálsorral tudjuk vizsgálni, de itt se felejtsük el, hogy egy e-mail címben kötelezően lennie kell egy @-nak és egy pontnak is. Javascriptben létezik külön e-mail cím validáló script, de az egyszerűség kedvéért php-ban írtam meg ezt a vizsgálatot.
4. Vizsgálnunk kell azt is, hogy ezzel a nicknévvel regisztráltak-e már. Ehhez kapcsolódnunk kell az adatbázishoz (mysql_query), és ott össze kell számolnunk mysql_num_rows függvénnyel, hogy létezik-e már az aktuális username (username=”. $adatok['nick].”1)”. Ha ez a szám nagyobb, mint nulla, akkor értesítsük a felhasználót, hogy válasszon másik nevet.
5. Végül ellenőrizzük, hogy ott, ahol meg kell adni a két jelszót, ott nem történt elírás. Ezt egy egyszerű egyenlőséggel vizsgálhatjuk meg, ahol az adatok tömb pass1-ét hasonlítjuk az adatok tömb pass2-jéhez. Az adatok tömbről és elemeiről még később szó lesz a regisztracio.php-ben.
6. Ha ez mind rendben van, akkor minden rendben, azaz az else ágban egy „rendben” szöveggel térünk vissza.

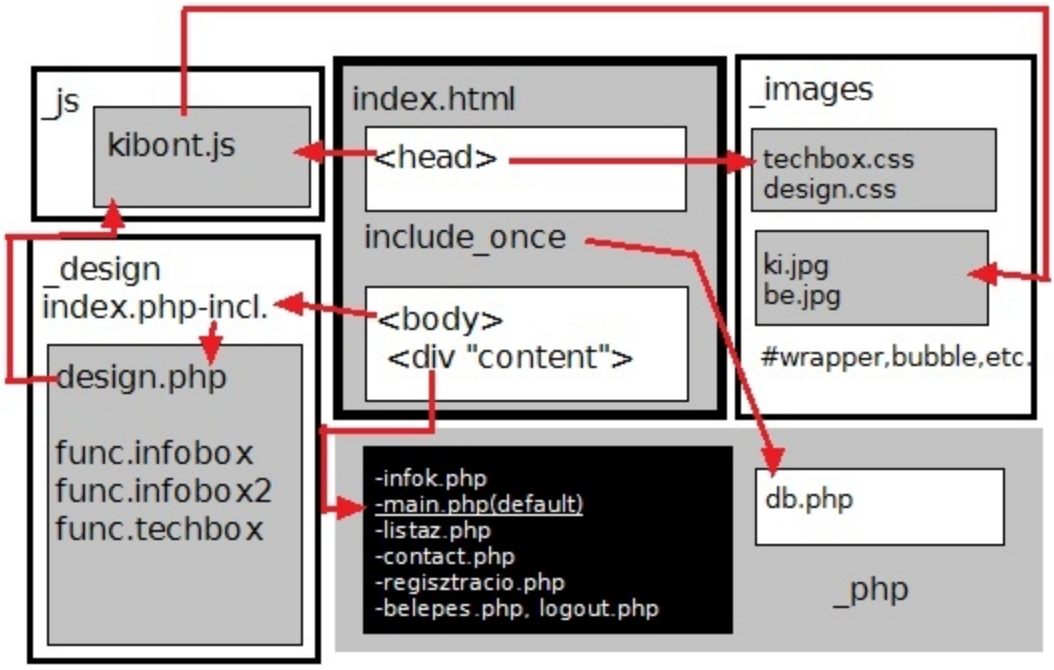
És végül a regisztracio.php függvényei és működése

A formai felépítés után, ahol text típusú cellákban bekértük az adatokat, jön egy submit, egy jóváhagyás. Ezután jön a funkcionalitása a regisztrációnak.

Első lépésben meghívjuk include-dal a már létrehozott reg_check.php-t. Megvizsgáljuk, hogy tényleg „leokézta” a user a regisztrációt, és ha igen, meghívjuk a reg_check-et erre a POST-ra. Ha rendben van minden („rendben”), akkor megnyitjuk az adatbázist, és insert into-val feltöltjük az új adatokat, amikre az „adatok” aktuális tömbbel tudunk majd később hivatkozni. Végül a jelszót, az emailt, a címet és a telefonszámot md5-be kódoljuk a biztonság miatt, és hogy illetéktelen személyek ne kaparintsák meg (például telemarketing céljából). Ha valami gond akadt az adatokkal, lépjünk ki az if ágból, és az else ágban írassuk ki a problémát:

print reg_check($_POST);

Ha nem tudunk kapcsolódni az adatbázishoz, azt is írassuk ki, így könnyebben rájövünk, mi a teendő hiba esetén. Egyéb esetben írjuk ki, hogy sikeres volt a regisztráció, és irányítsuk át a vezérlést az index.php-ra egy „tovább” segítségével.

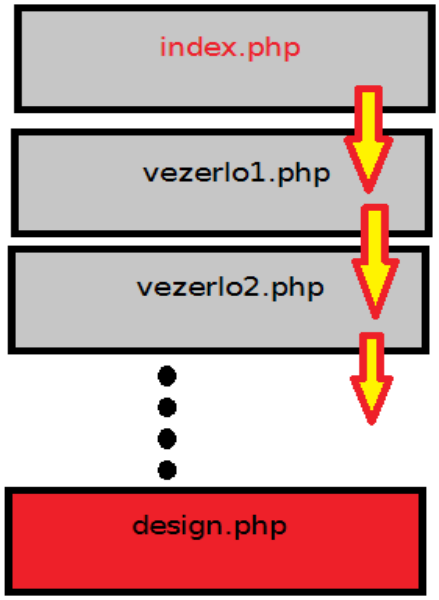


A paradoxonok beépítése a kódba

Készen van a nem funckionálisan működő webshopunk, most építsük be a két paradoxonunkat, a homonkulusz-paradoxont és a vezérlő egység paradoxont!

Kezdjük az egyszerűbbel, a vezérlő egység paradoxonnal. Azt tudjuk, hogy az

index.php-ban meghívódik a _design mappánkban az index.php, ami include-olja egyből a design.php-t. Azaz a vezérlés átkerül az index.php-ről egyből. De mi lenne, ha nem egyből kerülne át, hanem az index.php-ről include-dal átkerülne egy vezerlo.php-ra? Nézzük meg, hogy nézne ki a kódunk, ha ez háromszorosan is egymásra épülne:



include_once "./_design/vezerlo1.php"; //index.php átadja vezerlo1-nek

include_once "./_design/vezerlo2.php"; //vezerlo1.php átadja vezerlo2-nek

include_once "./_design/vezerlo3.php";//vezerlo2.php átadja vezerlo3-nak

...

include_once "./_design/design.php"; //vezerloN.php végül átadja design.php-nak, és vége.

Hogy mi ennek a jelentősége? Ha olyan programrészt vizsgálnánk, ahol minden egyes vezérlésnél újabb feltételre vizsgálnánk az eddig bekért kódot, akkor hiba esetén tudnánk, hol, melyik egységnél történt a hiba. Például ha a reg_check.php-t felbontanánk annyi részre, amennyi feltételt magában tartalmaz, akkor ahol hiba lenne, abban a php fájlban tudnánk olyan hibakezelést végezni, amivel egyrészt nem engednénk tovább a többi php felé, másrészt a többire nem vonatkozna az a kód. Erre jó példa, ha már a nick névből el tudjuk dönteni, hogy az éppen regisztráló egyéntől nem fogunk kérni bizonyos adatokat. Például lenne megkülönböztetve több szerepkör, úgymint csak nézelődő, csak offline boltban vásárló, és ténylegesen online vásárló ember is.

Lehetséges olyan eset is több adatbázis esetén, ahol a regisztráció során megadott

adatoktól függően kell eldöntenünk, melyik adatbázisba kerüljön be. Például ha az e-mail címben vizsgálnánk, hogy milyen nemzetiségű az emberünk (hu, jp, de, uk), akkor ő a neki megfelelő adatbázisba kerülne be. Ez akkor jó példa, ha a bolt beindulása után nem akarjuk megváltoztatni a létező adatbázisunkat (új oszlop „nemzetiség” néven), hanem inkább külön létrehozunk egy ugyanolyan táblát/adatbázist a külföldieknek is.

A vezérlő egységeket sohasem szabad túlzásba vinni! A sokszor felesleges vezérlési átadások növelik a várakozás és az adatbázis foglaltság idejét, ami egy cégnél jelentős bevételkiesést eredményezhet.

A homonkulusz-paradoxon kedvéért pedig nyissunk egy külön „Információk” fület, amit jól látható módon tegyünk feltűnővé. Mindig elérhető legyen, akár be vagyunk jelentkezve, akár nem.

5.A WebShop adatbázisa és táblái

Ahhoz, hogy egy webshopban le tudjuk tárolni a rendeléseket, a felhasználókat és magukat a termékeket, szükségünk van egy adatbázisra, amiben az adatainkat táblákban tárolhatjuk. Én ehhez a PhpMyAdmin nevezetű szoftvert használtam, ami gyors, átlátható, és egyszerűen kezelhető.

	Tábla	Művelet	Rekordok	Típus	Illesztés	Méret	Felülírás
☐	makett		8	MyISAM	utf8_hungarian_ci	2.4 KB	-
☐	rendeles		6	MyISAM	utf8_hungarian_ci	2.1 KB	-
☐	user		8	MyISAM	utf8_hungarian_ci	2.7 KB	-
	3 tábla	Összeg	22	MyISAM	utf8_unicode_ci	7.3 KB	0 Bájt

Az itt látható táblákban semmi újdonságot nem találhatunk, de felhívhatja figyelmünket magára a MyISAM típus. Hogy mi is az a MyISAM, megtudhatjuk, de előbb ismerkedjünk meg a z ISAM függvénnyel.

Mi az az ISAM?

Az ISAM egy gyors adatindexelést elősegítő metódus, teljes nevén Indexed Sequential Access Method, azaz az Indexelt Szekvenciális Hozzáférési Metódus. Az IBM

fejlesztette ki nagy mennyiségű adat feldolgozására és több, terminálokon keresztül kapcsolódó felhasználó kiszolgálására használt központi gépekre. A mai értelemben

többféle értelemben is használjuk:

- az IBM ISAM terméke, és az általa tartalmazott algoritmus
- egy adatbázis rendszer, melynél egy alkalmazásfejlesztő közvetlenül használ API-kat, hogy megadott sorrend alapján keressen indexeket, és találjon meg rekordokat az adatfájlokban. Ezzel ellentétben egy relációs adatbázis lekérdezés optimalizálással automatikusan készíti ezeket az indexeket.
- egy indexelő algoritmus, ami lehetővé tesz soros és kulcsos adathozzáférést. A legtöbb adatbázis a B-fa valamelyik változatát használja ugyanerre a célra, bár az eredeti IBM ISAM és a VSAM implementációk nem.
- általános értelemben, bármely adatbázishoz tartozó index. Manapság mindenféle adatbázis, akár relációs, akár más használ indexeket.

Egy ISAM-rendszerben az adatok fix hosszúságú rekordokba vannak rendezve. A rekordokat szekvenciálisan tárolják a szalagokon, eredetileg a gyorsaság növelésének érdekében. Egy másodlagos hash-táblákból álló halmaz segítségével, melyeket táblamutatóknak is hívunk, képesek vagyunk bármely rekordot lekérni, anélkül, hogy a teljes adatbázist be kellene járnunk. Ez annak a korábbi megoldásnak a megkerülése, amikor még a rekordokban helyezkedtek el a más adatokra mutató pointerek.

A kulcsfontosságú újítás az ISAM-ban, hogy az indexek kicsik és gyorsan kereshetővé teszik az adathalmazt, megengedve az adatbázisnak, hogy csak azokhoz a rekordokhoz férjen hozzá, amikre valóban szüksége van. Nem szükséges megváltoztatni más adatot, mikor mindenféle adatmódosítást hajtunk végre a kérdéses táblán és az indexeken.

Amikor létrejön egy ISAM fájl, és az indexek beállítódnak, akkor a mutatóik nem változnak a későbbi beszúrások és törlések esetén (csak a levélcsúcsok változnak ekkor). Ennek következményeképp, ha újabb levélcsúcsok kerülnek beszúrásra az adatbázisba, és azok meghaladják a tárolás maximális kapacitását, akkor egy úgynevezett "túlcsordulási" láncokon tárolódnak le. Ha a további beszúrások száma meghaladja a törlések számát, akkor ezek a túlcsordulási láncok hatalmasra nőhetnek, ami erősen meghosszabbítja a lekért rekord visszaadásának válaszidejét.

Táblák között létrehozott linkek helyességének fenntartását segítő, a plusz logikával ellátott ISAM-ba könnyen építhetőek relációs adatbázisok. Tipikusan a külső kulcs látja el ennek a linkként használt mezőnek a szerepét. Bár lassabb, mintha egyszerűen csak letárolnánk az érintett adat rekordmutatót, de így bármilyen fizikai változtatás az adaton nem vonja maga után a mutatók újradefiniálásának szükségességét, hisz még mindig érvényesek maradnak.

Az ISAM implementálása könnyen érthető, mivel elsődlegesen direkt, soros hozzáférést biztosít az adatbázis fájlhoz. Ezek mellett még hihetetlenül olcsó is.

Optimalizációnak nevezzük azt, amikor a kliens gépnek vezérelnie kell a saját kapcsolódását az egyes fájlokhoz. Ebből következik, hogy felmerül az inkonzesztencia esélye az adatbázisok állapotaiban, ha ugyanazon fájlokat akarjuk beszúrni. Ennek egy hagyományos megoldási módszere a kliens-szerver keretrendszer hozzáadása, ami felügyeli a kliens kéréseket és menedzseli a parancsokat. Ez az alap koncepciója a DBMS-nek, ami egy kliensréteget alakít ki az alatta lévő adattár rétegre.

Mi az a MyISAM?

A MyISAM a MySQL relációs adatbázis-kezelő rendszer 5.5-ös verziójának az elsődleges, alapértelmezett tárolási rendszere. A korábbi ISAM-kódon alapul, de rendelkezik sok hasznos újítással. A legfőbb változtatása a MyISAM -nek a tranzakciós támogatás teljes hiánya. Az 5.5-ös és magasabb verziójú MySQL-ek InnoDB-re változzak, hogy biztosítsák a hivatkozási integritási megszorításokat, és a magasabb konkurenciát.

Minden MyISAM tábla a lemezen három fájlban kerül letárolásra. A fájlok nevei a konvenció alapján a tábla nevével kezdődnek, kiterjesztésük pedig a fájl típusáról nyújt információt. Például az .frm kiterjesztés egy tábla definícióját tárolja, ám ez nem része a

MyISAM-nak, csak a szervernek. Az adatfájlok kiterjesztése .MYD (MYData), és az indexeké a .MYI (MYIndex).

Ezek után tekintsük meg, hogy például a termékeink milyen formában fognak letárolódni.

```
16 CREATE TABLE IF NOT EXISTS `makett` (
17   `m_id` int(11) NOT NULL AUTO_INCREMENT,
18   `m_nev` varchar(50) COLLATE utf8_hungarian_ci NOT NULL,
19   `m_leiras` varchar(50) COLLATE utf8_hungarian_ci DEFAULT NULL,
20   `m_ar` int(11) NOT NULL,
21   PRIMARY KEY (`m_id`)
22 ) ENGINE=MyISAM  DEFAULT CHARSET=utf8 COLLATE=utf8_hungarian_ci AUTO_INCREMENT=4 ;
23
24 INSERT INTO `makett` (`m_id`, `m_nev`, `m_leiras`, `m_ar`) VALUES
25 (1, 'Shermann-tank', '2.világháború, műanyag, 1 figura', 3400),
26 (2, 'Elf íjászok', 'Fantasy, műanyag, 20 figura', 4000),
27 (3, 'Tigris tank', '2.világháború, fém, 1 figura', 4100),
28 (4, 'Birodalmi alabárdosok', 'Fantasy, fém, 10 figura', 4500),
29 (5, 'Római légionárus doboz', 'Ókor, műanyag, 30 figura', 3000),
30 (6, 'Numídiai lovas doboz', 'Ókor, műanyag, 12 figura', 2400);
```

Termékek listázása php oldalra az adatbázis táblájából

Most hogy rendelkezünk a php oldal és az adatbázis tábláink összes fontos ismeretével, nézzük meg, hogyan tudunk php-ra kiíratni (pontosabban a Böngészés alfülbe, a lister.php nevezetű php kódba) adatbázisból adatokat. Természetesen itt is a mysql_query és mysql_db_query függvények lesznek a segítségünkre:

```
<?php
   if ($isLoggedIn)
   {     $dbname = "makettbolt";  /*adatbázis neve  */
         $dbuser = "root";
         $dbpass = "";

       $chandle = mysql_connect("localhost", $dbuser, $dbpass) or die("Nem sikerült
                csatlakozni az adatbázishoz.");
       mysql_select_db($dbname, $chandle) or die ($dbname . " A megadott adatbázis
nem     található. " . $dbuser);
      $mainsection="makett"; /*a tábla neve*/
```

```
$keresendo = ($_POST['termeknev']);
/*Ez volt a közös rész, amikor még nem tudjuk, a vevő minden terméket ki akar-e
        listázni, vagy csak egy bizonyosat.*/
//ha csak egy terméket akarunk megkeresni
if($_POST['termeknev'])
{
$query1="select * from ".$mainsection." where m_nev=". "\"".$keresendo. "\"";
        /* a keresendő termék benne van-e*/
 $result = mysql_db_query($dbname, $query1) or die("Nem sikerült a következő
        lekérdezés: " . $query1);
  /*hajtsa végre a lekérdezést*/
 $thisrow=mysql_fetch_row($result);
 if ($thisrow)  /*ha talált valamit a lekérdezés, és nem üresen tér vissza*/
 {
  echo "Megtaláltuk a keresett terméket.<br>";
  $eredmeny = mysql_query($query1);
  $sor = mysql_fetch_assoc($eredmeny);
  echo "<br>";
        print_r("Név: ".$sor[m_nev]." Azonosító: ".$sor[m_id]." Leírás: ".
$sor[m_leiras]."        Ár: ".$sor[m_ar]);      }
  else      {  echo "Nem találtuk meg a keresett terméket.<br>";   }
}
 else if ($_POST['osszestermek'] ) /*ha az összes termék nevét akarjuk látni*/
{
  echo "<br>";
  echo "Összes kapható termék: <br>";
  $query2="select m_nev from ".$mainsection."";/*minden termék*/
  $eredmeny2 = mysql_query($query2);
$result2 = mysql_db_query($dbname, $query2) or die("Nem sikerült a következő
    lekérdezés: " . $query2);  //hajtsa végre a lekérdezést
  $thisrow2=mysql_fetch_row($result2);
  if ($thisrow2)
  {  /*ha talált valamit a lekérdezés, és nem üresen tér vissza*/
    while($sor2 = mysql_fetch_assoc($eredmeny2))
    {/*így kell az ÖSSZES terméket kiírni*/
```

```
            echo "<br>";
            print_r("Név: ".$sor2[m_nev]);
          }       }       else          {  echo "Nincs semmilyen termék a boltban.<br>";
          }    }    }
    else
    {    print 'Nem vagy bejelentkezve!';}
```

Tehát vagy kilistázza a vevő az összes termék nevét, vagy pedig egyetlen egy termék nevére keres rá, és annak minden adatát látni fogja, azza a táblában a megadott névhez tartozó ($keresendo) termék összes attribútumát (azonosító, név, leírás, ár).

ÖSSZEFOGLALÁS

Paradoxonok az informatikában, akárcsak a matematikában, a fizikában, a közgazdaságtanban és rengeteg más területen az életben mindig voltak, és lesznek. Az ember gyakran összefuthat velük, és ha rendelkezik kellő ismeretekkel róluk, akkor megpróbálhatja elkerülni, esetleg megoldani őket (több-kevesebb sikerrel).

Dolgozatom elkészítése során törekedtem a könnyebb megérthetőségre, ezért ahol tudtam, példával szolgáltam egy-egy elméleti háttér gyakorlati bemutatására. Töredelmesen elnézést kérek azoktól, akiket egy igazi, funkcionalitásában minden téren lefedett WebShop elkészítésének ötlete foglalkoztatta, de ez nem is volt célom, hiszen a WebShop csak olyan célból jött létre, hogy a paradoxonokat egy könnyen érthető, mindennapos példában be tudjam mutatni. Ennek ellenére nem fog csalódást okozni annak sem a dolgozat, aki a PHP5 nyelvről szeretne olvasni, ugyanis rengeteg beépített függvény, eljárás és fogalom került megmagyarázásra a forráskódok bemutatásának részében.

Szomorúan tapasztaltam továbbá, hogy a paradoxonok témája hiába nyújt érdekfeszítő és emlékezetes tanulmány készítésére lehetőséget, sajnos magyar nyelvű, értékes forrást nehezen lehetett találni az informatikai szakterület témájában. A forrásaim ezért egytől egyig angol nyelvről lettek fordítva, amivel remélem megkönnyítem azon emberek tanulmányait, akik más idegen nyelven tanultak, de a téma iránt szintén érdeklődtek, és én is próbára tehettem szakfordítói képességeimet.

Irodalomjegyzék

1. *http://www.cs.auckland.ac.nz/~chaitin/amsci.pdf*
2. *http://www.szabilinux.hu/php/function.ob-start.html*
3. *http://php.net/manual/en/function.include-once.php*
4. *http://www.w3schools.com/php/php_get.asp*
5. *http://hu.wikipedia.org/wiki/Homunkulusz-paradoxon*
6. http://www.buzzle.com/editorials/12-24-2001-8369.asp
7. http://pcforum.hu/hirek/12516/Linux-ra+valt+az+orosz+kormanyzat.html
8. http://pcforum.hu/hirek/10940/Vista-ra+cserelik+vissza+a+Linuxokat+Becsben.html
9. http://pcforum.hu/hirek/11670/Befuccsolni+latszik+Oroszorszag+Linuxra+valtasa.html
10. http://www.sulinet.hu/tart/fncikk/Kacd/0/32215/index.html

Köszönetnyilvánítás

Köszönöm Fazekas Gábor tanár úrnak eme érdekes, nem mindennapi téma kidolgozásakor nyújtott segítségéért, szakértelméért és végtelen türelméért. Nélküle nem készülhetett volna el időben és kellő odafigyeléssel eme dolgozat.