Debreceni Egyetem
Informatikai Kar

ALKALMAZÁSFEJLESZTÉS XML TÁMOGATÁSSAL

Témavezető:
Dr. Adamkó Attila
Egyetemi tanársegéd

Készítette:
Nagy István Zoltán
Programtervező Informatikus BSc

Debrecen
2009

0. Tartalomjegyzék

1. Bevezetés

Jelen dolgozat egy matematikai jellegű alkalmazás kifejlesztéséhez kapcsolódik, amely XML dokumentumokat használhat bemenetként vagy kimenetként a felhasználó igényeinek megfelelően. Ez viszont nem azt jelenti, hogy csak XML fájlok segítségével történő kommunikációra képes, hanem azt, hogy van lehetősége a felhasználónak az elvégzendő műveletek megfogalmazására szöveges bemenet formájában is a felhasználói felületen, illetve ha úgy dönt, akkor élhet a kapott lehetőséggel és létrehozhat új XML dokumentumokat, vagy feldolgoztathat már meglevőeket.

Azért esett erre az alkalmazásra a választás, mert véleményem szerint a sok professzionális, viszont meglehetősen drága matematikai programcsomag mellett van létjogosultsága egy alacsonyabb funkcionalitású, ingyenes, vagy olcsón elérhető matematikai szoftvernek is. Ennek az alkalmazásnak nagy hasznát vehetik a felsőoktatásban matematikát vagy természettudományokat hallgató diákok, mivel számukra nem éri meg komolyabb matematikai szoftvercsomagokat megvásárolni. Ők ugyanis nem képesek az összes lehetőség kihasználására ezekben a szoftverekben, mivel az általuk megoldott problémák jóval egyszerűbbek nagy általánosságban, mint az ilyen szoftverek által megoldható feladatok. Ettől függetlenül természetesen a felsőoktatásban dolgozó oktatók is hasznát vehetik az alkalmazásnak, mivel ugyan nem képes szimbolikus integrálásra, vagy szélsőérték-számításra, mint egyes fizetős alkalmazások, ettől még egyszerűbb feladatok elvégzésében jobb lehet, mint egy egyszerű számológép, amelyet a beépített többletszolgáltatásoknak köszönhet, mint például az egyváltozós függvények rajzolása, vagy az akár 50000 számjegyig növelhető pontosság. Erre ugyanis nem minden számoló alkalmazás képes. A téma kiválasztásakor komoly segítséget nyújtott az is, hogy a most kifejlesztett alkalmazástól egy primitívebbet már sikerült korábban létrehoznom. Ebből fakadt az ötlet, miszerint az akkor még C nyelven létrehozott, nem megfelelően megtervezett alkalmazás funkcionalitását kibővítve, jobb tervek alapján igényesebb kezelőfelülettel létre lehetne hozni egy magasabb minőségi szintű programot. Ez az ötlet tartalmazta azt is, hogy célszerű lenne, ha el lehetne menteni a már begépelt kifejezéseket, illetve később meg lehetne őket nyitni feldolgozásra. Ehhez szükség volt valamilyen dokumentumra, amelyben a kifejezéseket tárolhatom. Ekkor jelent meg a komoly lehetőségeket nyújtó technológia, az XML használatának ötlete. Annak is köszönhető

a választás, hogy az XML hierarchikus struktúrájára könnyedén leképezhetőek a kifejezésfák, amelyek szintén hierarchikusak.

Az alkalmazás kifejlesztésekor főbb elvárások közé sorolom:

- a tetszőlegesen állítható pontosságot,
- a grafikus felhasználói felületet,
- a függvényrajzolás megvalósítását,
- a többnyelvűség támogatását és
- az XML dokumentumok használatának lehetőségét.

Az alkalmazás kifejlesztéséhez illetve a dolgozat megírásához használt szoftverek listája a következő sorokban olvashatóak:

- NetBeans IDE
- OpenOffice.org Writer
- Microsoft Windows XP
- Fedora 10

2. A program megvalósítása

2.1 A követelmények meghatározása

Az alkalmazás kifejlesztését megelőzően szükség volt az alapötletre, amelyről már az előzőekben volt szó. Az ötlet megfogalmazódásakor már ismert volt a követelmények egy része, de egyes elvárások csak a követelmények listájának összeállításakor, illetve az ezzel párhuzamosan zajló technológiai megvalósíthatóság vizsgálatakor derült fény. Az első követelmények a következőek voltak:

- Legyen képes az alkalmazás matematikai kifejezések minél szélesebb körének kiértékelésére.
- Legyen lehetőség XML fájlok használatára bemenetként és kimenetként egyaránt.
- A kifejezések megadása megvalósítható legyen szöveges formában is a felületen teljesen zárójelezett infix alakban.
- A kifejezés szöveges megadása történhessen prefix alakban.

Ezeken túl a lista összeállítása során felmerült az igény, hogy valami olyan pluszt tudjon nyújtani az alkalmazás egy egyszerű számológéppel szemben, amelyet a piacon található társai közül nem sok teljesít. Ezért jelentek meg a lista következő elemei:

- Meg lehessen adni a műveletek elvégzésének kívánt pontosságát, hogy akár a több ezer számjegyre pontosan is lehessen számításokat végezni.
- Legyen az alkalmazás többnyelvű, hogy a felhasználók szélesebb köre legyen képes a használatára.
- Legyen lehetőség az újabb nyelvek hozzáadására egyszerű szövegfájlok fordításával.
- Legyen elérhető az alkalmazás több operációs rendszeren is.
- Legyen egységes, platformfüggetlen megjelenésű grafikus felület, hogy minden felhasználó azonos felületen tudja használni az alkalmazás összes funkcióját.
- Egy megfelelő előképzettségű felhasználó legyen képes egy napos betanítás után az alkalmazás használatára.

Ezek az igények már részben világossá tették a felhasználandó szoftverek egy részét, és a többnyelvűséget is szükségessé tevő ok további igényeket ébresztett, de ezeket már a Java ismereteimre támaszkodva tudtam felmérni, hogy megvalósíthatóak-e. Az előbb említett ok az volt, hogy minél több emberhez eljuthasson a szoftver, és ne legyen gond a telepítés az

emberek többségének. Ezért a platformfüggetlenség még fontosabb lett, és a Java nyelv kiválasztása végleges döntés lett. Emellett felmerült az igény a funkciók bővítésére is. Az új követelmények a következőek:

- Az ismert műveletek felhasználásával legyen lehetőség kétdimenziós, egyváltozós függvények kirajzolására.
- A függvény kirajzolt képét le tudja menteni a háttértárra valamilyen képfájlba.
- Legyen telepítő szkript vagy grafikus telepítő alkalmazás a program telepítésének egyszerű megvalósításához.
- Legyen a teljes telepítő készlet 25 MB-nál kisebb, hogy egy átlagos 1 Mbps sebességű internetkapcsolattal is gyorsan le lehessen tölteni az internetről.
- Legyen az alkalmazás telepítéséhez szükséges hely kevesebb, mint 50 MB.

Ezekkel a követelmények listája teljes lett. A követelmények alapján felmértem a megvalósíthatóságukat, mint ahogy azt már fentebb említettem. A megvalósíthatóságot tekintve arra a következtetésre jutottam, hogy valamennyi követelmény elérhető és teljesíthető, így nem volt szükség kompromisszumokra vagy a követelmények fontosság szerinti rangsorolására a szoftverfejlesztés következő lépéseinek eléréséhez.

A rendszerrel szembeni elvárások alapján teljes mértékben világossá vált már a kezdetekkor, hogy a Java nyelv lesz valószínűleg a legalkalmasabb a feladat elvégzésére az általam ismert nyelvek közül, mivel a program platformfüggetlenségét nem tudtam volna egyébként garantálni. Ezen felül azért is jó választásnak tűnt a Java használata, mert nagyságrendekkel kényelmesebb volt számomra így az alkalmazás létrehozása, mint ha C-ben kezdtem volna el a munkát. Az egyik legjelentősebb előny például az XML bemenetek értelmezésénél illetve az XML kimenet létrehozásakor tapasztalható szerintem, mert a Java tudtommal jóval nagyobb támogatást biztosít ehhez.

2.2 A tervek

A tervek létrehozásához ki kellett választanom a szükséges eszközöket is, amelyeket használni szerettem volna a későbbiekben. Ezekről a technológiákról a dolgozat későbbi részeiben lesz szó, ezért most nem ismertetem őket részletesen. A Java nyelv választását már az előző részben indokoltam, de röviden összegezve a C és a Java közül kellett választanom és a Java mellett tettem le a voksomat a kényelmesebb fejlesztés reményében. Ezen felül mivel XML bemenetre és kimenetre is szükség volt a két alternatíva, a SAX és a DOM közül az utóbbit kellett előnyben részesítenem, mert a két megoldás közül a DOM használható XML dokumentumok létrehozására is, így igazán a SAX-nak nem volt annyi előnye a feldolgozás sebességében, illetve memóriaigényességében, amely ennek ellenére indokolta volna a SAX használatát. Emellett az is tény, hogy a dokumentum feldolgozása után terveim szerint kifejezésfát építek a kifejezésből és a fa felépülése után értékelem ki a kifejezést. Ez jóval közelebb áll a DOM által kínált lehetőséghez.

Az alkalmazás megvalósításához finom szemcsézettségű architektúrát választottam, mert a karbantarthatóságot, illetve az evolúció iránti igény megjelenésére való felkészülést tartottam fontosnak. A teljesítmény így bizonyára nem lett olyan nagy, mint amilyen egy durvább szemcsézettségű rendszer esetén lehetett volna, de ha a kiértékelés sebességét akartam volna maximalizálni, akkor már a Java is rossz döntés lehetett volna, tekintve, hogy a C programok gyorsabbak általánosságban. A feladat elvégzését tehát több kisebb osztályt illetve ezeknek az egyes funkciók szerinti csoportosításával moduláris felépítést terveztem megvalósítani. A fejlesztés további lépései is ezek köré a kisebb egységek köré csoportosulnak, mivel - mint azt már írtam - egy-egy csoport egy-egy funkció megvalósítását látja el. A szoftver kifejlesztése során evolúciós prototípus alapú fejlesztést választottam pontosan a funkciók szétválaszthatósága miatt, így ha egy funkcióval végeztem, és a tesztek is sikerrel jártak, akkor kezdődhetett a következő lehetőség hozzáadása, amely a követelmények változásával könnyedén képes tartani a lépést, illetve komoly előny, hogy minden egyes részfeladat elvégzése után lezajlik a teszt, amely segít felderíteni az olyan hibákat, amelyeket az új funkció hozzáadása előtt még nem tapasztaltunk, de az új prototípusban már megjelentek.

A kifejezések modellezéséhez szükség volt egy osztályhierarchiára, amelynek a

példányaiból fel lehet építeni a fát. Ehhez kellett egy ősosztály, amelynek a fa valamennyi eleme példánya, és a gyermekei, az operátorok osztályai, illetve az operandusok osztályai. Az adatmodell létrehozásához szükséges volt még továbbá egy olyan technológia alkalmazása vagy kidolgozása, amely lehetővé teszi a szokásosnál jóval nagyobb pontosságú számokkal való számításokat is. Ilyen eszköz van beépítve a nyelvbe is, a neve BigDecimal. Én mégsem emellett döntöttem, mert a BigDecimal típushoz a Java nem kínál beépített eszközöket a különböző komolyabb matematikai műveletek elvégzéséhez, mint például a trigonometrikus függvények vagy egy szám négyzetgyökének illetve köbgyökének kiszámítása. Ezért két lehetőség jelent meg, vagy kifejlesztem saját magam a hiányzó műveletek meghatározásához az eszközöket, vagy keresek egy újrafelhasználható eszközgyűjteményt, amelyet beépíthetek az alkalmazásomba. Végül az utóbbi mellett döntöttem, mert rábukkantam egy számomra szinte teljesen megfelelő eszközgyűjteményre, az Apfloat-ra. Erről is lesz szó a későbbiekben, ezért csak annyit említenék meg róla elöljáróban, hogy kész megoldást nyújt a fent említett problémára, mivel beépítve tartalmazza a BigDecimal osztályból hiányzó operátorokat, amelyekkel elvégezhető minden művelet, amely a java.lang.Math-ben is megtalálható. A választás azért is volt így kedvező, mert az újrafelhasználhatóságnak köszönhetően sikerült kiváltani meglehetősen sok fejlesztést, amely teljes mértékben feleslegessé vált így. A könyvtár sajnos nem volt alkalmas ennek ellenére sem minden általam elvárt művelet elvégzésére, mivel terveim közt szerepelt, hogy mátrixokkal is lehessen műveleteket végezni. Ehhez kisebb kiegészítést kellett megvalósítani az Apfloat eszközeinek felhasználásával. A tervekben mátrixok invertálása, szorzása, összeadása, kivonása, transzponálása, determinánsuk valamint normájuk meghatározása, illetve skaláris szorzás szerepelt.

Ezt követően meg kellett tervezni az XML dokumentumokat is, hogy egyértelmű legyen a bemeneti illetve kimeneti dokumentumok felépítése.

A bemenő adatok validálását XML Schema használatával végzem, ezért a dokumentumok felépítésének eldöntése utána definiálni kellett a hozzájuk tartozó .xsd dokumentumot is. Az XML szerkezet megtervezésénél két elterjedt nézet közül választhattam: az első, miszerint nem használok attribútumokat, a másik pedig az, hogy használom őket. Az alkalmazás a második változathoz áll közelebb, de ha lehet, akkor inkább egy speciális esetről van szó. Ez annak köszönhető, hogy a dokumentumokban használhatóak az attribútumok bizonyos információk rögzítésére, de a használatuk nem kötelező mindenhol. Így lehetnek

olyan dokumentumok is, amelyekben egyetlen egy attribútum sincs, míg a másik véglet, hogy lehetséges az attribútumok használata is. Ennek az az oka, hogy a szerkezetben az attribútumok csak a számítás pontosságának, a mátrixok méretének illetve az operátorok típusának megadására szolgálnak. A pontosság mindig csak a gyökérként funkcionáló „expression” címkéjű elem „precision” attribútumaként jelenhet meg. Az attribútum értéke pozitív egész típusú szám kell legyen. Az operátorok típusát az „operands” attribútumok adják meg, amelyek az adott operátorhoz tartoznak, és értékük szintén az előjel nélküli egészek közül kerül ki. Ezeknek az attribútumoknak nem minden esetben kötelező megjelennie, mivel például egyoperandusú operátorok esetén egyértelmű, hogy ha nincs megadva az ezt feltüntető attribútum, akkor is egy lesz az operandusok száma. Fontos ugyan az is, hogy ha fel van tüntetve az operandusok száma, akkor a például hozott esetben csak egy lehet az attribútum értéke, mert egyébként nem érvényes a kifejezés. A speciálisnak megbélyegzett halmazoperátorok esetén viszont kötelező feltüntetni az operandusok számát, hogy pontosan tudjuk, mennyi elem tartozik az adott operátorhoz, mivel ezeknél a kifejezéseknél az operandusok listája változó számosságú lehet. Az XML dokumentumokban, amelyeket az alkalmazás bemenetként illetve kimenetként kezel, lehetőség van hosszú illetve rövid címkék használatára akár keverten is. Ez azt jelenti, hogy minden operátornak illetve operandusnak meg van adva, hogy milyen címkeneveket használhatunk, ha azt akarjuk, hogy az alkalmazás felismerje őket. Erre a legtöbb esetben két módot nyújt az alkalmazás, az első a hosszú, amely a könnyebb érthetőség jegyében az adott operátor angol neve szóközök és speciális karakterek nélkül, mint például a maradékos osztásnál „divisionwithreminder”. A rövid címke, amely ennek a párja lehet egyszerűen „divr” amely a gyorsabb szerkeszthetőség miatt lett ilyen, hogy azok a felhasználók, akik már régebb óta használják az XML bemenetet, könnyebben és gyorsabban állíthassák össze a kifejezéseiket. Ezekből már látható az is, hogy egyértelműen emberi olvasásra is alkalmasak a dokumentumok, amelyeket a program használ. A kifejezések felépítése szinte triviális, mivel az operátorok gyermekei egészen egyszerűen az operandusai lesznek, és így egy fát alkotnak. A fa gyökere minden esetben a fentebb már említett „expression” nevű elem lesz.

A tervezéskor a szoftver számára az alábbi minimális rendszerkövetelményeket határoztam meg, amelyek teljesülése esetén már képesnek kell lennie elvégezni a kapott feladatokat. Fontos megjegyezni természetesen, hogy a felsorolásban olvasható hardverek

esetén a műveletvégzés bonyolult kifejezésekkel és nagy pontossággal lassú lehet, ezért célszerű a nagyobb feladatok elvégzéséhez jobb hardvert biztosítani.
A követelmények:

- 250 MHz-es AMD vagy Intel processzor
- Minimum 128 MB memória
- Minimum 10 MB szabad terület a merevlemezen
- Videokártya
- Minimum 1024x768-as felbontású képernyő
- Java Runtime Environment 6
- Microsoft Windows XP/Vista/2003/2008 vagy Linux

Úgy érzem ezek nem jelentenek nagyon nagy elvárásokat, így az alkalmazás szinte minden mai korszerű számítógépen képes ellátni a feladatát. A leginkább kritikus pont a teljesítménnyel kapcsolatban a processzor lehet, ezért ajánlott minél jobb központi egységgel felszerelt gépen futtatni a programot.

A terv szerint az implementáció során használt eszközök listája az alábbi:

- Java
- Apfloat
- NetBeans IDE
- XML
- XML Schema
- DOM
- JUnit
- i18n
- Swing

A felhasznált eszközök meghatározását az algoritmus illetve a tesztesetek megtervezése követte, majd csak ezután kezdődhetett az implementáció. Az algoritmus meghatározása funkciónként külön zajlott természetesen az előző funkció eredményeinek felhasználásával. Emiatt ezekről a tervekről az implementációs részben lesz szó.

2.3 Az implementáció

Az implementáció illetve a tervezés közvetlenül azt megelőző része, amelynek során meghatároztam a program előállításához szükséges algoritmust, valamint a tesztek megtervezése már az egyes funkciók szerinti felbontást követte. Minden funkció előtt megterveztem, hogy hogyan szeretném hozzáadni az újnak számító funkcionalitást, illetve milyen tesztek sikeres elvégzése esetén tekintem megfelelőnek az alkalmazást. Az evolúciós prototípusok használata azért előnyös, mert így minden funkció hozzáadásával létrejön egy olyan alkalmazás, amely már az eredeti feladatok egy részét meg tudja oldani. Ez viszont pontosan a hátránya is ennek a módszernek, hiszen a megrendelő bármikor azt mondhatja, hogy meggondolta magát és nincs szüksége a maradék funkciókra, inkább fejeződjön be a fejlesztési folyamat. Ettől a hátránytól itt nem kellett tartanom, mivel nem tartottam valószínűnek azt, hogy nem kell befejeznem a dolgozatot. A funkciók listája, amelyeket egyesével meg kellett valósítani a következő sorokban olvasható:

- A mátrixokkal kapcsolatos műveletek megírása Apfloat típusra
- A kifejezések kiértékelése
- XML dokumentum érvényesítése és beolvasása
- Prefix kifejezésből XML generálása
- Infix műveletekből XML generálása
- Grafikus felület illetve a vezérlője
- Függvények rajzolása
- Nemzetköziesítés megvalósítása
- A megoldás XML-be exportálása
- Telepítőprogram megírása

Az fentebb leírtakból nem szeretnék minden részletre kitérni, mivel ebben az esetben túlságosan nagy terjedelmű lenne a dolgozat, de a fontosabb elemekről természetesen szót ejtek az alábbiakban.

2.3.1 Kifejezések kiértékelése

A lista második elemeként szerepel a kifejezések kiértékelése pont. Ehhez meg kellett terveznem egy adatmodellt, amelyet már ezelőtt is említettem. Ez egy kifejezésfa, amelynek minden eleme egy-egy objektum, amelyek a kifejezésfa elemeinek ősosztályából, az

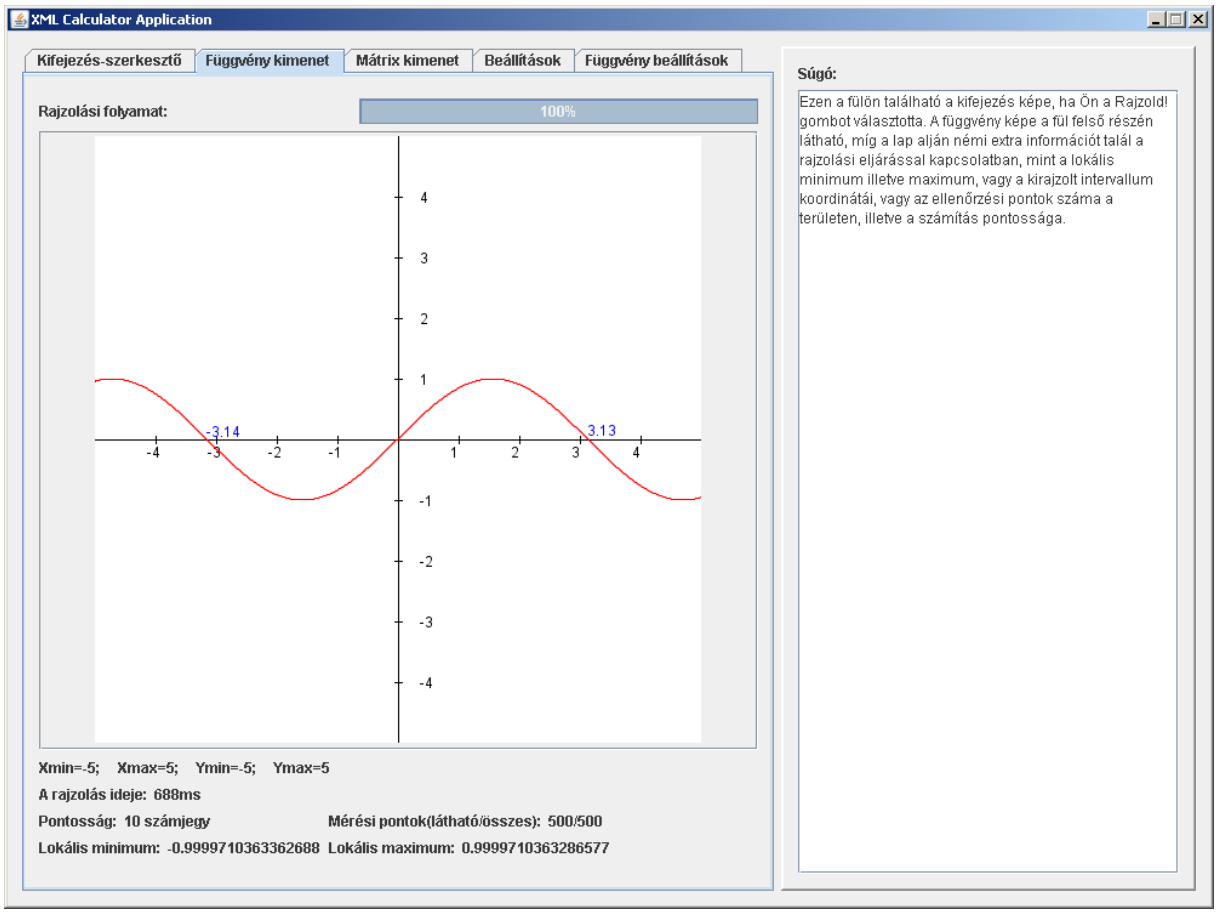
„ExpressionTreeElement”-ből származik. A fa felépítéséhez felhasznált összes osztály az „expressiontree” csomagban található. A kifejezés kiértékelésének módja úgy történik, hogy minden osztály örökölt egy absztrakt metódust a fa elemeinek ősétől, amely arra szolgál, hogy az adott elem értékét meghatározza. Ez a getValue() metódus, amely egy új faelemet ad vissza a hívás eredményéül. Ezt a metódust kellett minden osztályban megírni, hogy az egyes operátorosztályok példányai meg tudják mondani az értéküket a gyermekeiktől megkapott érték alapján. Ennek köszönhetően csupán a gyökérként nyilvántartott elemtől kell megkérdezni az értékét, hogy a kifejezés kiértékelődjön. Ez persze nem minden kifejezés esetén értelmes, mert például a mátrixok szorzásánál elég komoly megszorítások vannak a művelet elvégezhetőségére. Ezért szükség volt kivételosztályok létrehozására is, hogy a hibákat a saját kivételek kiváltásával jelezhessem. A tesztek tervezésénél szinte minden esetben JUnit tesztek megtervezéséről van szó. Ez azt jelentette ennél a modulnál, hogy minden osztályhoz létre kellett hozni egy tesztet, amely több általam ismert eredményű értékkel is elvégezte a teszteket, és a tesztek eredményeit összevetette a várt értékkel. Ezeknek a tesztosztályoknak a létrehozásában nagy segítséget jelentett a NetBeans IDE által nyújtott támogatás, amellyel pár kattintással eljuthatunk abba az állapotba, hogy csak meg kell adnunk a konkrét teszteseteket és már mehet is a tesztelés.

2.3.2 Függvényrajzolás

A számomra leginkább kedves funkció a függvényrajzolás volt. Bár a megvalósítás során voltak gondok az algoritmusokkal, illetve nem volt könnyű feladat, hogy a mostani formáját elérje, de úgy érzem, hogy ez egy hálás feladat volt, és megérte a fáradságot. Ezért ezt a pontot is a kiemelten fontos funkciók közé emelem. A függvények megvalósítására vonatkozó követelmények azt mondták meg, hogy olyan függvényeket kell tudnia kiértékelnie a programnak, amelyek egyváltozósak és csak olyan operátorok vannak a függvényben, amilyeneket a program ismer. Ennek köszönhetően szükség volt egy olyan elemre, amely a változó szerepét tölti be a kifejezésfában, illetve az XML dokumentumban. Erre már a tervezés egy korábbi fázisában is fény derült, ezért a NumberType osztályban, amely egyébként a számok, mint operandusok megjelenítésére szolgál, elhelyeztem egy logikai értéket, amely hamis ha egyszerű számról van szó, és igaz ha változóról. Ennek köszönhetően a fa már fel volt készítve a változók fogadására, míg az XML dokumentumokban a VAR

értékű „number” elem szimbolizálja a változó helyét. Az algoritmus megalkotása így viszonylag egyszerűen történhetett. A végleges megoldásban a rajzolást végző osztály megkapja a kifejezést, amiből fát épít az eddigi eszközök felhasználásával, majd megkeresi az összes változót a fában és ezeket egy egyirányú láncolt listára is felfűzi, hogy gyorsan el lehessen érni ezeket az elemeket. Ezt követően beállítja a rajzolás tulajdonságait, mint a kirajzolandó intervallumok kezdeti és végpontjai illetve a számítás minőségére vonatkozó pontossági illetve az iteráció lépéseinek számát eldöntő kritériumok. A következő lépés a koordináta-tengelyek berajzolása kell legyen, amelyre speciális megoldást kell alkalmazni, hogy a beosztások ne legyenek túl sűrűek. Ez könnyen megoldható a kirajzolandó tartomány méretének felmérésével, illetve egy-egy többirányú elágazás beiktatásával. Ezt követően értékül adja a változók láncolt listájának minden elemének a vízszintes tengelyen megadott intervallum kezdőpontját. A kifejezés kiértékelését követően az eredményt jelentő pont koordinátái átesnek egy úgy nevezett window to viewport transzformáción. Ez a számítás annyiszor zajlik le egyenlő lépésközökkel a kirajzolandó intervallum elejétől a végéig, amennyi az iteráció lépésszáma. A szomszédos pontokat egyenes vonalakkal összekötve folytonos görbét kapunk, amely már közelít a függvény valós képéhez. Arra kell csak ezen felül odafigyelni, hogy ha szakadást tapasztalunk a függvényben, azaz van valamilyen olyan eset, hogy a látható tartományon kívülre esik az egyik pont, majd a közvetlenül őt követő szintén, de már nem azon az oldalán a tartománynak, azaz az első fölötte van a területnek, a második pedig alatta, akkor a két pont közti egyenest nem szabad berajzolni, mert csak egy fölösleges függőleges vonalat kapnánk. Ha a függvény nem értelmezett az adott mérési pontban szereplő koordinátaértékre, akkor nem rajzolunk oda semmit. Ezt kivételekkel tudjuk érzékelni. Az iteráció során feljegyezzük a legkisebb és a legnagyobb értékeket, mint a függvény becsült lokális minimumát illetve maximumát, valamint összeszámoljuk a kirajzolt pontokat is. Ezek az értékek azért lesznek fontosak a későbbiekben, mert ha a felhasználó olyan kifejezést ad meg, amelynek egyetlen pontja sem esik a mérési tartományba, akkor a program a minimum és a maximum értékek ismeretében képes beállítani az optimális határokat a függőleges tengelyen értelmezett intervallumra vonatkozólag. Ez lehetővé teszi, hogy a felhasználó dönthet a függvény újrarajzolásáról az új értékekkel, vagy beérheti pusztán azzal az információval is, hogy az adott területen nincsenek pontok. Az ehhez szükséges becsült lokális szélsőértékekkel illetve a kirajzolt pontok számával kiegészíthető a

függvényrajzolásról szóló jelentés is a felületen, így akár a felhasználó ezeket is felhasználhatja a munkája során.



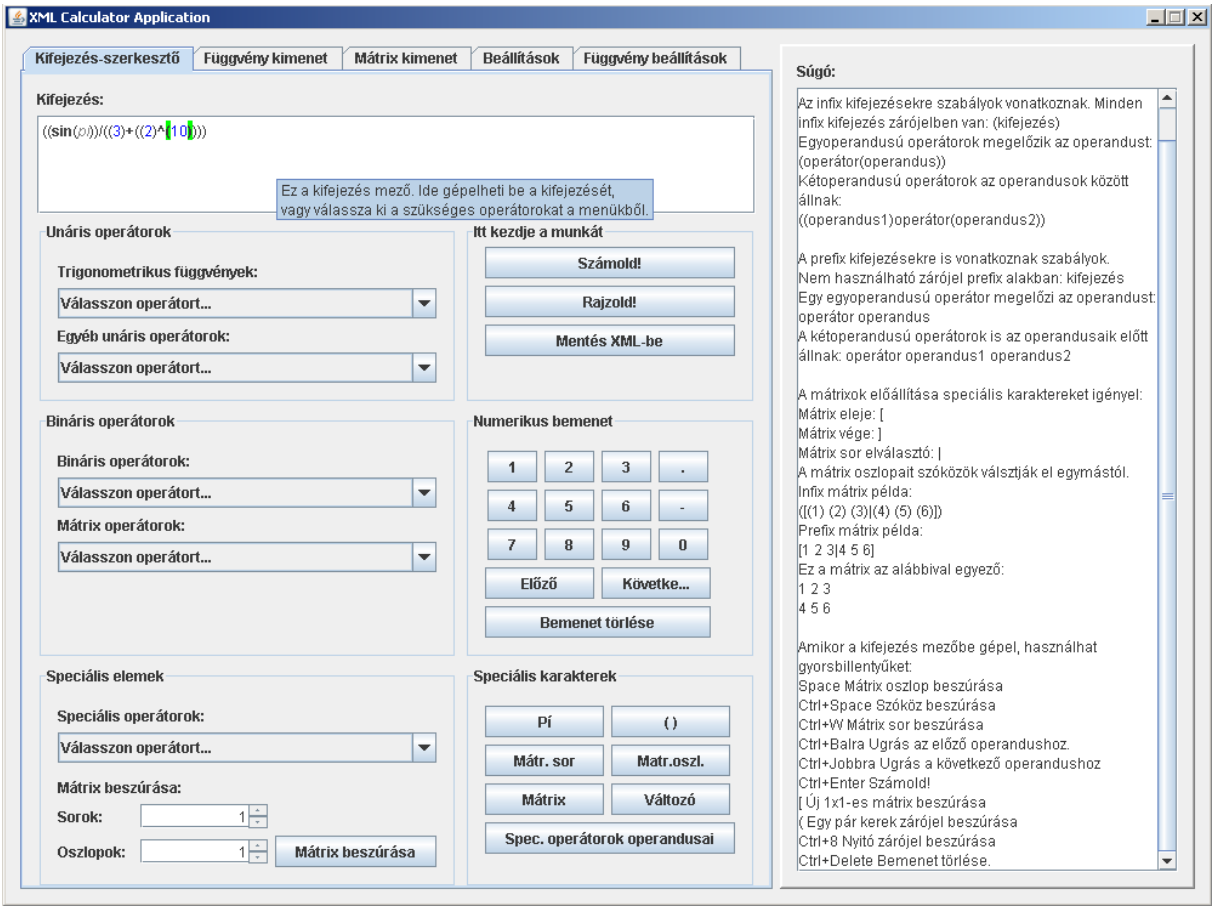
1. ábra: A sin(X) függvény kirajzolásának eredménye.

A szakadásokkal rendelkező függvények pontosabb képe meghatározható úgy is, hogy amikor szakadást tapasztalunk, akkor a két oldalán található pontok között további részekre osztjuk az intervallumot, hogy az új pontokban is meghatározhassuk a függvényértékeket. Az iteráció során alkalmaztam még egy feltételes utasítást, amely a felhasználó kérésének megfelelően bejelöli a zérushelyeket illetve a függőleges tengely és a függvény metszéspontját, majd kiírja a közelítő értékét ezeknek a pontoknak a rajzvászonra. Ennek az algoritmusnak a tesztelése nehezen automatizálható lett volna, illetve a képek összehasonlítását egyszerűbbnek láttam, ha a saját szememmel teszem meg, ezért manuálisan végeztem el a teszteket.

2.3.3 Felhasználói felület és nemzetköziesítés

A felhasználói felület megtervezése volt az egyik legnehezebb feladat. Eldobható prototípusokkal próbáltam a használhatóság illetve a felhasználóbarát felület felé haladni. A grafikus felület megvalósításához a Swing-et választottam, mert a Swing felületek jellemzője, hogy minden operációs rendszeren képesek azonos megjelenés elérésére, így teljesült az erre vonatkozó követelmény. Az alkalmazás által ismert sok funkció és operátor nagyon bonyolult grafikus felületet eredményezett volna, ha minden egyetlen oldalon jelent volna meg gombok formájában. Ezért rá voltam kényszerítve a JTabbedPane nevű eszköz használatára, amely lehetővé tette, hogy több fülön csoportosítsam a szükséges elemeket. Az első fülön kaptak helyet a szöveges kifejezések megszerkesztéséhez szükséges eszközök. Ezek viszont olyannyira soknak bizonyultak, hogy ha gombokat alkalmaztam volna, nem fértek volna el a képernyőn. Ezért csoportosítottam az operátorokat az egyes típusok szerint és listákba rendeztem őket. A kifejezések bonyolult szintaxisa miatt nem lett volna egyszerű egy összetett kifejezés megszerkesztése minden segítség nélkül, ezért a felület tervezésekor kidolgoztam egy megoldást, amelynek a segítségével kiszínezhető a bemenet, és a felhasználó azonnal látja a színekről, hogy helyes, vagy nem helyes az éppen begépelt operátor neve, vagy megvannak a nyitó zárójelek párjai. Ennek köszönhetően a kifejezések szerkesztése, különös tekintettel a teljesen zárójelezett infix kifejezésekére jelentősen leegyszerűsödött. A kifejezések szerkesztésére természetesen nem csak egy mód áll rendelkezésre, hanem lehetőség van a számunkra legmegfelelőbb lehetőség választására. Az első lehetőség a kifejezés egyszerű begépelése haladó felhasználóknak, amelyhez az előzőekben említett szintaktikai kiemelés tartozik.

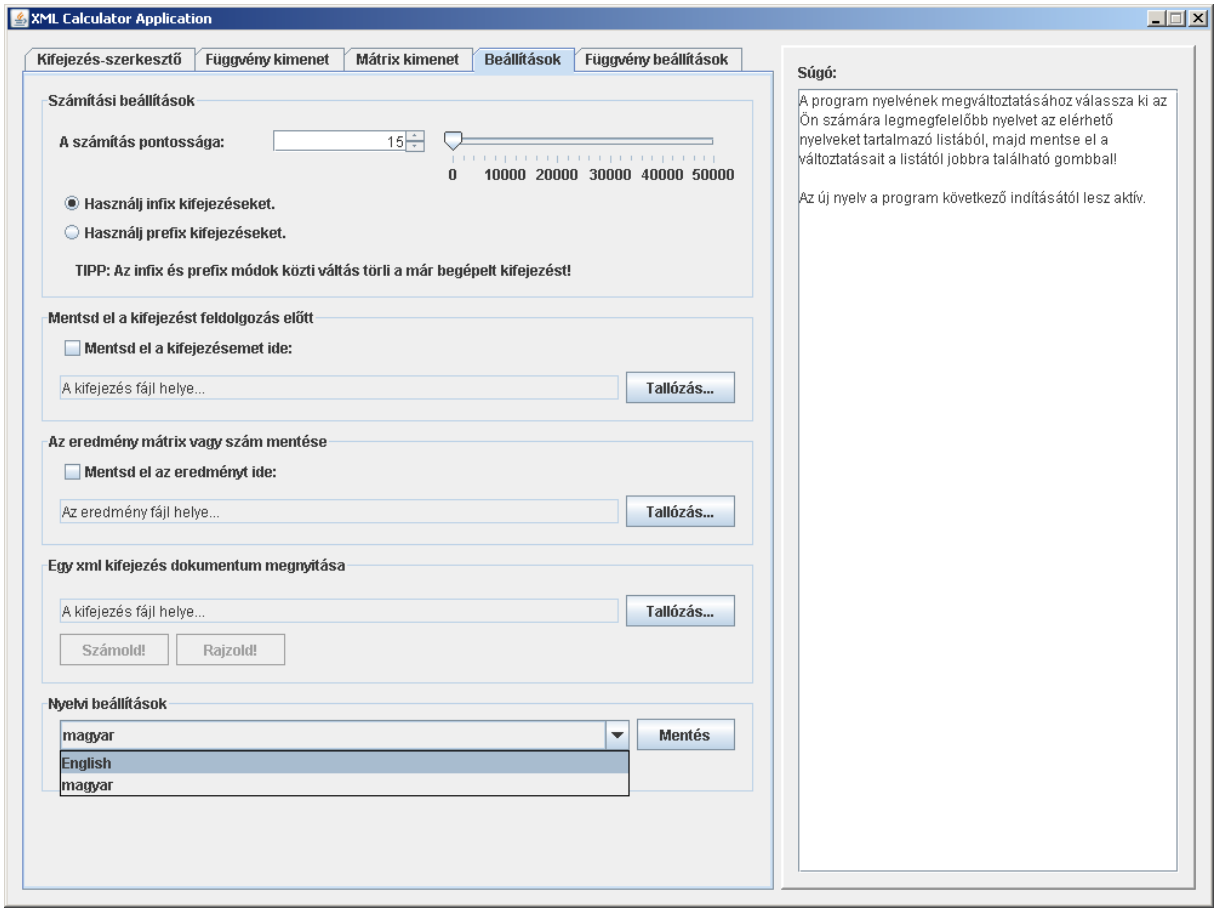
A legutolsó prototípusba már bekerült pár billentyűkombináció is, amelyek a gépelés felgyorsítására szolgálnak. Ilyenek például a zárójelek automatikus párban történő beszúrása, illetve az ennek elkerülésére szolgáló alternatíva, vagy a mátrixok megadását könnyítő billentyűkombinációk, amelyek új elem; illetve sorelválasztó karakter és új elem beszúrására is használhatóak. A másik megoldás a kifejezésnek a fentebb említett listák illetve a speciális jelek és számok gombjainak használatával történő összeállítása. A szintaktikai kiemelés, illetve a kifejezések szerkesztésére használható felület a 2. ábrán tekinthető meg.



2. ábra: A kifejezések szerkesztésére alkalmas fül és a szintaktikai kiemelés.

A kimenet megjelenítése két fület vesz igénybe, mivel a kifejezések kiértékelése és a függvényrajzolás eredménye is nagy helyet vehet igénybe. Ezeken felül két fül szükséges a beállítások megadására, illetve az XML fájlokkal való műveletek megvalósítására. A felület jobb oldalán egy állandóan látható szövegmező van, amelyben az egér pozíciójának követésével helyzetérzékeny súgószöveg található. A felület tervezésekor figyelembe kellett venni, hogy nem egy, hanem több nyelvet kell támogatnia az alkalmazásnak. Ehhez minden szöveget, ami a felületen megjelenik nyelvfájlokban kellett letárolni, és onnan az aktuális beállításnak megfelelőt beolvasni. A nemzetköziesítés feladatát több módon is megoldhattam volna, de én azt választottam, hogy ResourceBundle segítségével valósítom meg a több nyelv használatát. Ezt a funkciót tovább szerettem volna javítani azzal, hogy kidolgoztam egy megoldást, amely támogatja a nyelvek utólagos hozzáadását. Ezt úgy terveztem meg, hogy a program indításakor számba veszem az elérhető összes olyan nyelvet, amelyet a JVM támogat, majd ezeknek a listájával töltöm fel a nyelv kiválasztására szolgáló listát. Első

indításkor angol a nyelv alapértelmezetten, majd ha ezt a felhasználó átállítja valamelyik másik elérhetőre, akkor a következő alkalommal már a kiválasztott nyelven kommunikál velünk az alkalmazás.



3. ábra: Az alkalmazás nyelvének kiválasztása.

A felület tesztelése is manuálisan, emberi erővel történt, hiszen a gépek nem tudják eldönteni az olyan fontosságú tulajdonságokat, mint például az ergonómia illetve a könnyű használhatóság. Az ilyen tulajdonságokat gyakran a programozók sem tudják helyesen megítélni, ezért célszerű őket szakemberekkel megtervezetni. Az említett teszteknek köszönhetően sikerült kifejleszteni a végleges felületet.

2.3.4 XML kifejezések

Az XML szerepe központi az alkalmazásban, ezért ezt a funkciót mindenképpen be akartam illeszteni a dolgozatba. Mivel tudtam, hogy az XML lesz a menthető illetve beolvasható dokumentumok formátuma, a bemenő adatok érvényesítésére nem fektettem

nagy hangsúlyt a szöveges bemenet esetén, hanem XML dokumentumot hozok létre a kapott kifejezésből, majd azt XML Schema ellenében érvényesítem és felépítem a kifejezésfát. Ez azzal az előnnyel jár, hogy az XML dokumentumok beolvasásakor használt módszerekkel tudtam validálni az egyébként bonyolult, nehezen érvényesíthető szöveges matematikai kifejezéseket. Ennek köszönhetően az alkalmazásnak csak XML adatokból kell tudnia számolni, és nem kell külön módszerekkel elérni, hogy a prefix illetve a teljesen zárójelezett infix kifejezések használatához saját algoritmust konstruáljak. Az előnyök viszont hátrányokkal is járnak, miszerint a dokumentum legyártása, majd feldolgozása plusz idő, amely lassítja a feldolgozást. Szerencsére ez a többletidő a másodperc töredékével arányos, így a felhasználó észre sem veszi a különbséget. Az XML adatokkal végzett tesztekhez létrehoztam több XML dokumentumot is, amelyeknek pontosan ismertem a bennük tárolt kifejezés értékét, illetve olyanokat is, amelyek nem érvényesek, és mindegyikre lefuttattam a feldolgozást, majd a kifejezések kiértékelését. Amikor már ezek a tesztek megfelelő eredménnyel zárultak akkor lehetőség volt a szövegből generált XML dokumentumok beolvasására is, amelyeknek a segítségével az új kifejezések létrehozását is sikeresen tesztelhettem.

Az egyes funkciók a rendszerbe történő beillesztése után minden esetben szükségét láttam az egész rendszer tesztelését együtt is, hogy kizárjam a modulok egymásra gyakorolt hatásából adódó hibákat is.

2.4 A szoftver életciklusai a tesztek után

Az előállított alkalmazás vonzatában még nem igazán tudok a szoftver utóéletéről nyilatkozni, mivel még igazság szerint egyetlen egy felhasználója sem akadt aki napi munkájában hasznosítani szeretné a szoftvert. Ennek köszönhetően a karbantartás illetve a felhasználói visszajelzések feldolgozása nem kezdődött el. Ezt meg kell előznie még a bevezetés fázisának.

A szoftver bevezetését támogatandó létrehoztam a felhasználói kézikönyvet, amely a program beszerzésétől a munka elkezdéséig nyújtott segítségen túl megpróbálja megismertetni az alapvető használat módját is a felhasználóval. Ennek a kézikönyvnek a teljes szövege megtalálható a függelékben.

Ahhoz, hogy egyáltalán eljussunk a munka elkezdéséhez szükséges, hogy fel tudjuk telepíteni az alkalmazást a számítógépünkre. Az egyszerűség kedvéért létrehoztam egy olyan saját készítésű telepítőprogramot, amely minden platformon képes futni, amelyen maga az alkalmazás is, illetve pár egyszerű kattintás megtételével használható állapotba hozza az alkalmazást. A telepítő program is több nyelven érhető el, amely azt jelenti, hogy a lehetséges nyelvek közül a számítógépre telepített JVM alapértelmezett nyelvén fog kommunikálni a felhasználóval, vagy ha ez nem található, akkor angolul. Ezt azért tartottam fontosnak létrehozni, hogy a végfelhasználó informatikai hozzáértése vagy annak hiánya ne okozzon gondot a telepítés során sem, illetve a több nyelvnek köszönhetően sokkal kényelmesebben használható a telepítő.

A karbantartás illetve a szoftver evolúciója könnyedén támogatható a finom szemcsézettségű rendszerből adódóan. Kis objektumokat ugyanis sokkal könnyebb adott esetben cserélni, mint a nagyobbakat. Az evolúciót tekintve új funkciók hozzáadására is létezik mód, mivel a fejlesztés maga is evolúciós prototípusokkal zajlott. Ennek köszönhetően amellett, hogy a mostani verzió már használható, az új dolgok hozzáadásának irányában folyhat a követelmények feltárása vagy a módosításokhoz szükséges tervezés is.

A szoftver üzemen kívül helyezésére is létezik egyszerű megoldás, amelyhez a felhasználói kézikönyv is segítséget nyújt. Ha a programot el szeretnénk távolítani, akkor csupán törölnünk kell pár könyvtárat, és ezzel sikeresen üzemen kívül tudtuk helyezni.

3. Felhasznált technológiák

3.1 Bevezetés

A dolgozat ezen szakaszában fogom ismertetni az általam felhasznált technológiákat, illetve azokat a tulajdonságaikat, amelyeket a legfontosabbnak találtam megemlíteni. Meg kell jegyeznem, hogy a dolgozatnak nem célja a technológiák mélyreható ismertetése, csupán a főbb tulajdonságaikat, illetve az egyes technológiákhoz kapcsolódó alapvető információkat szerettem volna feltüntetni, hogy az olvasó megismerhesse a dolgozatban előforduló, gyakran angol nyelvű rövidítések jelentését, illetve azt, hogy mire használhatóak az egyes eszközök, vagy mi az, ami jellemzi őket. Ennek köszönhetően az alkalmazás kifejlesztésének során hozott döntéseim indoklása is szilárdabb lábakon állhat. Betekintést nyerhetünk a szoftver működéséhez felhasznált elemek egy csoportjába, azaz megtekinthetjük, hogy mi mozgatja a program képzeletbeli fogaskerekeit.

3.2 XML

Az XML az Extensible Markup Language, magyarul Kiterjeszthető Jelölő Nyelv rövidítése. Az XML használata mára széles körben elterjedt a leíró dokumentumoktól a SOA illetve az RSS 2.0 technológiákig. Ennek köszönhetően elmondhatjuk, hogy a mai felhasználók jelentős része használt már XML-t, csak mivel nem látták, hogy milyen folyamatok zajlanak a háttérben, nem is szereztek róla tudomást. Véleményem szerint az XML-ről legrövidebben az XML tíz pontjának áttekintésével kaphatunk információkat. Ezek a következők.

1. Az XML célja az adatok strukturálása

Az XML jó alapul szolgálhat az ember és a számítógép közti kommunikációra. A számítógép számára könnyen generálható, de az ember számára is egyértelmű. Az XML használatához, illetve a dokumentumok létrehozásához nem szükséges, hogy programozó legyen valaki.

2. Az XML egy kicsit olyan, mint a HTML

Az már első ránézésre is szembetűnő, hogy szerkezetét tekintve egy HTML dokumentum és egy XML dokumentum hasonlóan néz ki. Mindkettőben < és > jelek között jelenik meg a markup és az elemeket lezáró jelölések is ugyanolyan </ és > jelek között található szöveggel vannak megoldva. Ez persze még nem jelenti azt, hogy a két technológia ugyanaz lenne, de tény, hogy az XML sokat köszönhet a webböngészők fejlődésének és fordítva. A jelentős különbséget az XML-t jellemző szigorúbb kötöttségek jelentik, amelyek a HTML-ben nem feltétlenül léteznek. Gondoljunk csak az XML-ben megkövetelt dokumentumot indító sorra, amely megmondja, hogy milyen XML verziót használunk, vagy gyakran a dokumentum által használt karakterkódolást is. Emellett a HTML-ben felhasználható címkeneveket DTD-ben definiálták, így nem fordulhatnak elő más nevek, csak az előre megadottak. Ennek köszönhető az is, hogy a Leíró nyelvek által meghatározandó négy kritériumból a HTML teljesíti az utolsót, azaz meg tudjuk mondani, hogy mit jelentenek a markup-ok, míg az XML-nél ezt nem tehetjük meg, mivel ott teljesen személyre szabottan választhatjuk ki az általunk használt elemneveket. A HTML-hez közelebbi XHTML-ről a 7. pontban lesz szó.

3. Az XML szöveg, de nem olvasgatásra való

Ez a pont kicsit igaz is, és hamis is, mert vannak leíró XML dokumentumok, amelyeket igenis olvasgatásra találtak ki, viszont a gépi feldolgozásra szánt dokumentumok esetén elmondhatjuk, hogy nem emberi olvasásra készültek, ezért nem is valószínű, hogy értelmezhető lesz a dokumentum emberek számára, mivel a gépek szempontjából jelentős előnyként könyvelhető el, ha tömörebb dokumentumokat dolgoztatunk fel velük.

4. Az XML bőbeszédű, mert ilyennek tervezték

Az XML valóban bőbeszédű, mivel az információt jelentősen felduzzasztjuk például a címkék szövegével, vagy a záró elemek hossza is pazarlásnak tűnhet egy egyszerűbb adatszerkezettel szemben, viszont ennek ellenére jelentős előnyt jelent az XML számára, hogy strukturált, könnyen értelmezhető és hierarchikus. Ha a legrövidebb formát keressük az információ leírására, akkor nem az XML-t fogjuk használni, hanem valamilyen bináris ábrázolását az általunk leírandó információnak, ha viszont az XML mellett tettük le a voksunkat, akkor lehetőségünk van bizonyos esetekben némi takarékoskodásra a tárterülettel. Ezt a minél rövidebb címkék használatával tehetjük meg, mivel nem kötött a címkeneveket kell használnunk, hanem mi választhatjuk meg őket. Természetesen nem mindegy, hogy az adott dokumentumot emberek is olvasgathatják, vagy csak a számítógép fogja feldolgozni, mert ha csak gépi feldolgozás lehetséges, akkor célszerűbb sokkal rövidebb címkéket használni, mint az emberi olvasásra szánt dokumentumoknál, mert a gépnek úgyis teljesen mindegy, hogy milyen hosszú mintát kell felismernie, és minél rövidebb a címke, annál rövidebb a feldolgozásra vagy a hálózati kommunikációra fordított idő, amelyet ez a dokumentum igényel.

5. Az XML technológiák egy egész családja

Úgy érzem ez a kijelentés maximálisan megállja a helyét, mivel az XML feldolgozásához kínált technológia-paletta nagyon is széleskörű. Csak egy egyszerű esetet tekintve, ha egyetlen érvényes XML fájlt szeretnénk feldolgozni, akkor már a parser-ek közül is választhatunk, mivel adottak a DOM és a SAX feldolgozók is. Ehhez még választunk egy érvényesítési technológiát, amely akár a mára már kissé idejét múlt DTD is lehet, de célszerűbb az XML Schema, a Schematron illetve a Relax NG technológiák közül választani, akár több lépcsős érvényesítési technológiát is. Ezeket használva már nagy valószínűséggel

megjelenhetnek az XML névterek, illetve akár az XML transzformációs megoldások iránti igényeink is. Ehhez a sorhoz még több technológia lenne fűzhető bonyolultabb feladatok elvégzése esetén, mint például az XPath vagy az XQuery, amelyek az XML adatbázisok használatakor kerülnek előtérbe.

6. Az XML új, de nem előzmények nélkül való

Az XML őseként az SGML tekinthető, amely a Standard Generalized Markup Language, azaz Általános Szabványos Jelölő Nyelv rövidítése. Az SGML 1986-ban jelent meg nemzetközi szabványként a szöveges dokumentumok strukturális és tartalmi meghatározására. Már ez is validálható DTD-vel, amit az XML-ről is elmondhatunk, bár a DTD használata XML-lel ma már nem jellemző. Az XML fejlesztői lényegében átvették az SGML legjobb részeit és ehhez adták hozzá a HTML-lel kapcsolatos tapasztalataikat. Ennek köszönhetően az XML nem lett kevésbé hatékony, mint az SGML, viszont a használata jelentősen egyszerűbb lett.

7. A HTML-től XML-en keresztül vezet az út az XHTML-ig

Az XHTML nyelv tulajdonképpen olyan, mintha a HTML címkéket XML dokumentumokban használnánk. Ez azért lényeges, mert a HTML-hez hasonló dokumentumok jöhetnek létre, viszont a HTML-től eltérően az XHTML dokumentumoknak az XML-ből megismert jólformáltságnak is teljesülnie kell. Ez azt jelenti, hogy XML alapú dokumentumokat hozunk létre, csak az XHTML definíciója megmondja azt is, hogy mit jelent a markup, mivel ezt már a HTML is megteszi, és a címkenevek HTML-szerűek. Természetesen vannak eltérések a HTML és az XHTML között, de ezek nem igazán jelentenek nagy különbségeket.

8. Az XML moduláris

Az XML modularitása leginkább azzal indokolható szerintem, hogy például egy XML dokumentumnak nem szükséges egy fájlban elhelyezkednie, mivel van lehetőség áthivatkozásokra. Ennek a modularitásnak köszönhetően jelentősen javul a belső újrafelhasználhatóság. A hivatkozásokra is több mód van. Gondoljunk itt például az XLink és az XPointer technológiákra, amelyek az egyszerű egyirányú hivatkozások mellett képesek kétirányú illetve több célpontú linkek létrehozására is. A modularitás másik jó példája a

névterek használata, melyeket szabadon kombinálhatunk is, hogy egy új névteret hozzunk létre, amely már le tudja írni az általunk létrehozott adatstruktúrát.

9. Az XML az RDF és a Szemantikus Web alapja

Az RDF a Resource Description Framework, azaz Erőforrásleíró Keretrendszer rövidítése. Segítségével a nevéből is adódóan erőforrások leírására nyílik lehetőség. „Miként a HTML dokumentumokat, menürendszereket és formulákat tartalmazó applikációkat összekapcsolva elindította az eredeti Webet, úgy az RDF alkalmazásokat és ügynökprogramokat fog egy egységes Szemantikus Webbé formálni.” Az RDF-en alapuló, illetve a Szemantikus Web felé irányuló kutatások nagy hasznát vehetik például az internetes keresők, amelyeknek jelentősen javulhat a találati relevanciája, ha pontosan meg tudják mondani, hogy mi az, ami egy oldalon található.

10. Az XML licencmentes, platform-független és sokak által támogatott

Mivel az XML szöveg, platformfüggetlen. Ez abból adódik, hogy minden platformon lehet szövegeket létrehozni illetve feldolgozni, tehát egy új XML dokumentum létrehozása sem jelenthet problémát. A támogatottsága valószínűleg éppen a licencmentességének köszönhető részben, mivel ennek köszönhetően bárki bármikor használhat XML-t teljesen ingyen. Emellett az XML a W3C ajánlásaként ipari szabványnak számít. Ez a szabványosság pedig bizonyára vonzó a fejlesztők számára. A támogatottsághoz tartozik még az is, hogy a böngészők fejlődésével is rengeteg technológia került felszínre, amelynek köszönhetően az XML feldolgozása, használata is javult. Az XML egyik legnagyobb előnye épp a támogatottságában rejlik. Gondoljunk csak egy XML dokumentum feldolgozásához szükséges kódrészletre, ha java technológiát szeretnénk alkalmazni: minden technológiából találunk implementációt, amit használnunk kell a művelethez, nekünk csak a saját értéket kell hozzáadni, azaz megmondani, hogy mit akarunk az adatokkal elérni. (http://www.w3c.hu/forditasok/XML_10_pontban.html [2009.03.10. 19:01])
(http://www.w3.org/XML/1999/XML-in-10-points.html [2009.04.27. 11:00])

3.3 Swing

A Swing egy grafikus kezelőfelületek előállítására alkalmas könyvtár, amely a grafikus felhasználói felületekkel rendelkező programok előállításának megkönnyítésére hivatott. A grafikus felhasználói felületek létrehozására alkalmas még az AWT (Abstract Window Toolkit), azaz az absztrakt ablak eszköztár, de a Swing rendelkezik néhány plusz elemmel, amelyek hasznosak lehetnek. Talán ezek miatt lehet a Swing manapság egyre népszerűbb. Fontos, hogy ez a két csomag nem helyettesíthető egymással, nem jelentenek alternatívát. Bizonyos esetekben akár mindkét osztálykönyvtár használatára szükség lehet, mivel az AWT is rendelkezik olyan elemekkel a felületek elrendezéseire illetve az események figyelésére vonatkozóan, amelyeket a Swing nem tud megoldani, még hasonló eszköz sem található rájuk a benne, viszont az AWT sem rendelkezik például minden Swing vezérlőelemmel, mint például a jelszavak bevitelénél használatos JPasswordField, az előrehaladás jelzésére szolgáló JProgressBar, vagy a JToolbar nevű segédeszközsávok megvalósítására szolgáló osztályok. A Swing elemei visszanyúlnak az osztályhierarchiában a java.awt.Component-ig, bár a háttérben álló koncepció nem teljesen egyezik az AWT mögött állóval.
(D. Louis - P.Müller [2006] 214-218.oldal)

A Swing egységességet kínál a program megjelenését tekintve minden platformon, míg az AWT az adott platformon megszokott grafikus felhasználói elemeknek megfelelő külsőt biztosít a szoftver számára. Ez a különbség jelenthet előnyt és hátrányt is mindkét technológia számára. Az AWT például egységesebb képet biztosít az operációs rendszeren futó többi alkalmazással, viszont az egyes rendszerek közötti különbségeket figyelembe kell vennie, és ez komoly hátrány lehet a használhatóság szempontjából. Emellett negatívum, hogy az AWT-ben vannak kisebb következetlenségek, mint a címkék beállítására használt setText() illetve setLabel() metódusok kevert használata, vagy a nem teljesen azonos konvenciót alkalmazó osztálynevek írásmódja. A Swing ezzel szemben átgondoltabb, például mindig a setText() használata az elfogadott. További előnye a Swing-nek, hogy az AWT fejlesztése már befejeződött, míg a Swing evolúciója még folyamatban van. Ha pedig minden áron az operációs rendszer által használt megjelenést szeretnénk a programunkon viszontlátni, akkor megtehetjük azt is, hogy a Swing felületre „rákényszerítjük” a kívánt stílust.
(D. Louis - P.Müller [2006] 214-218.oldal)

3.4 Apfloat

Az Apfloat csomag a www.apfloat.org webhelyen fellelhető matematikai eszközgyűjtemény Java illetve C++ nyelvekhez. Az általam kifejlesztett alkalmazás a Java verziót használja, azon belül is az 1.5-öset. „Az Apfloat egy nagy teljesítményű Tetszőleges pontosságú aritmetikai könyvtár. Több millió számjegyes pontossággal végezhetünk számításokat a segítségével. Ugyanolyan egyszerű használni, mint a Java BigDecimal vagy BigInteger osztályai, de sokkal jobban teljesít extrém pontosságú számokkal (több, mint egy pár száz számjegy). Emellett egy teljes csoportja elérhető a matematikai függvényeknek tetszőleges pontosságú számokhoz: az összes java.lang.Math-ben elérhető és továbbiak. Az Apfloat a GNU Lesser General Public License (2.1-es verzió vagy bármely későbbi) alatt érhető el, és NINCSEN GARANCIA rá.”
(http://apfloat.org/apfloat_java/ [2009.04.27. 11:26])

Az Apfloat fejlesztője Mikko Tommila. Ezek az eszközök csak Java 5-ös vagy annál frissebb verzióval működnek. Az Apfloat használatának jelentős előnyét a BigDecimal illetve BigInteger osztályok használatával szemben a fentebb említett matematikai függvények implementációja jelenti, mivel nem szükséges saját magunknak például szinusz, koszinusz vagy más trigonometrikus függvények közelítését végeznünk, csupán importálnunk kell az org.apfloat.ApfloatMath eszközeit, majd használhatjuk az előre elkészített eszközöket minden plusz teher nélkül, mintha csak a java.lang.Math-ben található metódusokat használnánk persze némi szintaktikai eltérést tekintve, mivel itt nem egyszerű double értékekről van szó, hanem például az org.apfloat.Apfloat osztály példányairól. Az Apfloat osztályból példányosított objektumok képesek továbbá pár hasznos jellemzőjük megmondására is, mint például a double vagy long típusra átszámított értékük megadására, vagy az általuk ábrázolt szám többféle sztringen történő ábrázolására is.

3.5 DOM

A DOM rövidítés a Document Object Model angol szavak rövidítése, amely magyarra Dokumentum Objektum Modellként fordítható. A szavak jelentését a következő definíció segítségével szeretném feloldani:

„A Dokumentum Objektum Modell egy platform,- és nyelv-semleges interfész, amely lehetővé teszi a programok és szkriptek számára, hogy dinamikusan hozzáférjenek és módosítsák a dokumentumok tartalmát, struktúráját és stílusát. A dokumentumok további feldolgozása után a feldolgozás eredménye megjeleníthető a prezentált oldalon.” A DOM tehát dokumentumok feldolgozására és szerkesztésére is használható. Tulajdonképpen egy API érvényes HTML és jólformált XML dokumentumok feldolgozására és létrehozására.

A DOM-nak több szintjéről beszélhetünk, amelyek hosszú évek alatt alakultak ki:

A nulladik szintként a Legacy DOM tekinthető, amely 1996-ban kezdett kialakulni az 1990-es évek úgynevezett böngészőháborújának egyik eredményeként. Elsőként a Netscape Navigator 2.0 illetve az Internet Explorer 3.0-ban jelent meg valamilyen megoldás ezen a téren, bár ezek még nem feltétlenül voltak egységesek, inkább egy olyan megoldásról beszélhetünk, mint ami igen gyakran előfordul ma is, hogy ha a konkurenciának van egy megoldása valamire, akkor nekünk is kell legyen, tehát kidolgoztak valamit mind a két cégnél. Itt még semmilyen megkötés nincs. 1997-re megjelent az úgynevezett Intermediate DOM, amely már CSS támogatással is rendelkezett a Netscape Navigator 4.0 illetve az Internet Explorer 4.0 nevű böngészőkben.

Az első szint egy standardizálási folyamat eredményeként állt elő 1998-ban a W3C ajánlásaként. A W3C az ECMAScript (a böngésző szkript nyelvek egy standardja) létrehozásának érdekében a Netscape Communications és a Microsoft, valamint több egyéb böngészőgyártó bevonásával próbálkozott. Amikor az ECMAScript megszületett, nekiláttak egy standardizált DOM létrehozásának. Ez a DOM Level 1 néven hivatkozott szint, amelyben már a teljes HTML modell készen volt.

2000-ben jelent meg a második szint, DOM Level 2, amelynek az eseménymodell illetve az XML névterek támogatása jelentették az újdonságait.

2004-től beszélhetünk a harmadik szintről, amelyet DOM Level 3 néven hivatkozhatunk. Ebben már támogatják az XPath-t, a billentyűzet-események feldolgozását és

az XML-be történő szerializációt is.

A DOM a dokumentumot, amelyet manipulál, legyen szó feldolgozásról, vagy módosításról, egy fa szerkezetben ábrázolja, azaz egy fát épít belőle. Ez a fa csomópontokból áll, amelyeket angolul Node objektumoknak hívnak. A fa felépítése gyakran költséges lehet, mivel itt a dokumentumok egészét, vagy legalább az eddig már érintett részét a memóriában kell tartanunk. Cserébe viszont képesek vagyunk a dokumentum módosítására is, amely komoly előnynek számít szerintem. A DOM képességeit tekintve tökéletes választás lehet XML adatokkal dolgozó szoftverek fejlesztésére, mivel képes szinte mindenre, amire szükség lehet:

- dokumentumok létrehozása, felépítése
- dokumentumok szerkezetének bejárása
- elemek illetve tartalom hozzáadására, szerkesztésére és törlésére

A DOM API két megoldást biztosít a dokumentumok feldolgozására. Az első egy öröklődési hierarchián alapuló OO megközelítés, míg a második egy egyszerűbb nézetet nyújt ami azt vallja, hogy minden csomópont.

A DOM feldolgozás úgy zajlik, hogy az elemző (parser) létrehozza az XML dokumentumból az elemzett dokumentumot (parsed document), majd az elemzett dokumentumból létrejön a DOM fa, amelyet az alkalmazás felhasználhat bemenetként, vagy módosíthatja is.

A DOM nagy előnye természetesen a módosítás lehetősége, viszont meg kell említeni a gyors elérést is, amit a DOM biztosít.

Hátrányként könyvelhető el viszont a nagy memóriaigény, és a SAX-hoz képest alacsony teljesítmény.
(http://www.w3.org/DOM/ [2009.04.29. 20:05])

3.6 SAX

A SAX a Simple API for XML szavakból származó rövidítés, amelyet kissé körülményes magyarra fordítani, mivel az API és XML szavak magyarul is értelmesek szerintem. A magyar verzió Egyszerű API XML-hez lehetne esetleg. A SAX egy közkedvelt alternatívát nyújt a DOM technológiával szemben. A SAX a feldolgozás során eseményvezérelt megoldást alkalmaz callback-ekkel. Nem épít teljes elemzési fát, mint a DOM, csak a dokumentum végigolvasása során talált eseményeket, mint például a nyitó, vagy a záró elemek olvasása, kezeli le a hozzájuk tartozó kezelőmetódusok meghívásával. Ennek köszönhetően sokkal kisebb memóriaigénnyel rendelkezik, mint a DOM. Az is igaz viszont, hogy sokkal nagyobb teljesítményre képes. Sajnos a SAX nem képes mindenre, amire a DOM, inkább csak a dokumentumok végigolvasására nyújt megoldást, de még erre sem minden esetben alkalmas, mivel nem minden feldolgozás végezhető el vele. Ilyen eset például, ha belső referenciákat használunk. A SAX segítségével nem érhető el mindegyik elem folyamatosan. Ez egy komoly hátrány a DOM-mal szemben, mivel a DOM fában minden elem elérhető, csak a megfelelő irányban kell keresnünk. A SAX-nak nincs formális definíciója, hanem mindig a Java verzió a mérvadó. A többi nyelvre történő migrálás során a Java verzió szabályait kell követni, persze a nyelvek közti különbségek figyelembe vételével.

A feldolgozás során a SAX az alábbi eseményekkel kell megbirkózzon:

- XML elem csomópontok
- XML szöveges csomópontok
- XML feldolgozási utasítások
- XML megjegyzések

A fentebb már említett előnyök és hátrányok mérlegelése során mindenki eldöntheti, hogy DOM, vagy SAX elemzőt fog használni, de ha a funkciókat tekintjük a legfontosabbnak, akkor a DOM első ránézésre jobb választásnak tűnik. Ha viszont belegondolunk, hogy akár nagyon nagy fák is keletkezhetnek egy-egy méretesebb dokumentum olvasása során, és tudjuk, hogy csak beolvasni szeretnénk az információt, akkor a SAX mellett érdemes döntenünk.

(http://www.saxproject.org/quickstart.html [2009.04.29. 21:12])

3.7 XML Schema

Az XML Schema egy a jólformált XML dokumentumok érvényesítésére szolgáló eszköz, amelynek segítségével az XML dokumentum érvényességére vonatkozó megkötéseinket egy XML dokumentumban írhatjuk le. A Schema konkurenciájának tekinthető a Relax NG, illetve meg lehet még említeni a mára már kissé idejét múlt DTD-t is. Egy jólformált XML dokumentum akkor tekinthető érvényesnek, ha a logikai felépítése és a tartalma teljes mértékben megfelel a dokumentumban leírt, vagy külső dokumentumban leírt és az XML dokumentumhoz csatolt megszorításoknak. Az érvényesítési folyamatot az XML-feldolgozó végzi. Bemenete az XML dokumentum és a séma, kimenete pedig a validációs jelentés és opcionálisan a PSVI, azaz a Post-Schema-Validating-Infoset. Külön érdekesség, hogy ha XML dokumentumokkal dolgozunk, akkor azt nem kötelező érvényesíteni is, de ha érvényes dokumentumokkal dolgozunk, akkor a rendszert gyakran komoly problémáktól óvhatjuk meg, mivel ez egy lehetőség a bemeneti adatok ellenőrzésére. Egy jól megfogalmazott Schema definíció ezért igen hasznos lehet. Az XML Schema objektumorientált szemléletű, szemben a szabály alapú Schematron-nal, illetve a nyelvtanalapú nyelvekkel, mint például a Relax NG. Fontos megemlíteni, hogy az érvényesítési folyamat nem kötelezően egy lépésben zajlik, hanem akár több lépcsős megoldásokat is használhatunk, ahol kombináljuk az egyes technikákat, például ha az XML Schema és a Schematron együttes használatával szeretnénk megmondani, hogy az adott dokumentum érvényes-e, akkor amit az egyik nyelvvel nehéz, vagy nem megoldható ellenőrizni, azt a másikkal lehet, hogy könnyedén véghez tudjuk vinni. A Schema például nem képes megoldani, hogy a ha valaki azt jelölte be, hogy férfi, akkor férfi kell legyen a neve alapján is. Ezért a fentebb említett kombinált megoldáshoz hasonló technikákra van szükség ha a dokumentumokat minden szempont szerint validálni szeretnénk. Ezek a szempontok az alábbiak:

- szerkezet: az elemek és attribútumok szerkezete
- tartalom: a szöveges csomópontok és attribútumok tartalma
- integritás: egyediségvizsgálat, hivatkozások épsége
- üzleti szabályok: például a nettó ár az ÁFA mértéke és a bruttó ár közti különbség, vagy a helyesírás

Az XML Schema segítségével a DTD-nél jóval könnyebben hozhatunk létre szigorúbb szabályokat a dokumentum érvényesítésére. A Schema ugyanis tartalmaz típusokat is, amelyek megkönnyítik a munkánkat, illetve a beépített 19 primitív típuson felül alkalmazhatunk bonyolultabb adatszerkezeteket is, mivel akár saját típusokat is létrehozhatunk. Az ehhez biztosított eszközök között találhatjuk például

- a választólistát (Choice),
- a kötött elemsorrendű felsorolást (Sequence),
- a kötetlen elemsorrendű felsorolást (All),
- a számosságot befolyásoló minOccurs illetve maxOccurs attribútumokat.

Ezek az elemek nem minden esetben ágyazhatóak egymásba, például a choice illetve sequence nem állhatnak all gyermekeként. Az XML Schema talán legnagyobb nehézsége a specifikáció mérete. A Schema elsajátítása ugyanis szerintem azért nehéz, mert a specifikáció túl nagy, és hosszú idő kell, mire megérti valaki a felhasználáshoz szükséges információáradatot, míg a DTD például egy pár lépésből álló tutorial végigböngészése és néhány perc ráfordítása után használható. Természetesen ez annak is köszönhető, hogy a DTD sokkal kevesebb dologra képes, illetve rengeteg eszköz hiányzik belőle, ami a Schema-ban megtalálható.

3.8 Java

A Java egy objektumorientált programozási nyelv, amely szintaxisát tekintve a C és a C++ nyelvekhez hasonló, viszont a C++-ban találhatónál jóval egyszerűbb a Java objektummodellje. A Java fejlesztője a Sun Microsystems a 90-es évektől napjainkig. Igaz, most már az Oracle a Sun Microsystems tulajdonosa, de valószínűleg ők sem szeretnék abbahagyni a Java fejlesztését, ezért a Java a még ma is fejlődő nyelvek közé tartozik.

Fontos tudni a Java-ról, hogy bár elvileg minden platformon képes futni a speciális bájtkódja (bytecode) a mobiltelefonoktól a Lego robotokon át a számítógépekig, még ma is csak kevés olyan eszköz van, amely képes a Java bájtkód natív értelmezésére. Ezek közül egy a Sun Microsystems egy nem túl régi fejlesztése, a Sun Spot. A többi eszközön szükség van egy JVM nevű programra, amely képes lefuttatni a bájtkódot. A JVM rövidítés a Java Virtual Machine angol szavakat rövidíti, amelyet magyarul Java Virtuális Gép névvel illethetünk. A Java és a platformfüggetlenség kifejezések elég gyakran fordulnak elő együtt, mivel egy megfelelően megírt Java program (amely nem használ platformfüggő eszközöket, vagy mindig az adott platformhoz igazodik) képes lefutni például több operációs rendszeren is. Az egyik jelentős tény, amit meg kell említeni, hogy a JVM-ek ugyan képesek lefuttatni a kódot, de nem minden platformon ugyanolyan gyorsan történik a program lefuttatása. Ez annak köszönhető, hogy a JVM, ami lefuttatja a bájtkódot, egy szabványos felületet biztosít a futó program számára, így az működhet, csak ennek a felületnek a biztosítása előfordulhat, hogy a különböző platformokon több vagy kevesebb erőforrást igényelhet. Mindemellett vannak olyan virtuális gépek, amelyek csak a Java eszközök egy töredékét tartalmazzák, mint például a mobil eszközök programozására használt Java Mobile Edition különböző verziói. A Java ezeknek a kisebb problémáknak az ellenére valóban platformfüggetlen, mivel minden fentebb említett eszközön futnak Java programok, csak legfeljebb a be és kimenet illetve az eszközök platformfüggő megoldásainak a Java kódon belüli használata lehet nehezebb, ha hordozható kódot szeretnénk alkotni. Emellett természetesen az is igaz, ami a Java platformfüggetlenségének az igazi bizonyítéka is lehetne, hogy legyen szó például bármilyen operációs rendszerről, ha figyeltünk a programozáskor a rendszerfüggő részek megírásának módjára, akkor ha lefordítjuk az egyik operációs rendszeren a forrást, és az előállított bájtkódot átvisszük a másik operációs rendszerre, akkor az arra telepített JVM le fogja tudni

futtatni a programunkat minden komolyabb nehézség nélkül. Ennek köszönhetően ha egyszer megírunk egy Java alkalmazást, akkor azt több helyen is használhatjuk, így megspórolhatjuk a platformok közti különbségek elfedésére, vagy az új platformon történő implementációra és tesztelésre fordított erőforrásokat, legyen szó pénzről vagy energiáról. Innen ered a „Write once, run anywhere.” vagy néhol „Write once, run everywhere.” formában megjelenő szlogen, amely magyarul „Írd meg egyszer, futtasd bárhol.” formában hangozhat. Ezt a mondatot a Sun Microsystems használta a Java platformfüggetlenségének népszerűsítésére.

A másik fontos tulajdonság, amelyet a Java nyelvvel kapcsolatban szerintem feltétlenül meg kell említeni, az az újrafelhasználhatóság, amelyhez komoly támogatást kapunk a szoftverek fejlesztésekor. Ilyen eszközök lehetnek például teljes osztályok, vagy csomagok is, amelyeket egyszer valaki vagy valakik kifejlesztettek, hogy egy bizonyos problémát meg tudjanak velük oldani, de később egy hasonló feladat elvégzésekor ismét felhasználhatóak lesznek. Példaként lehet mondjuk hozni egy .jar fájlt, amely tartalmaz egy csomagot, amelyben minden osztály le van implementálva ahhoz, hogy a mátrixokkal végezhető műveletek elvégezhetőek legyenek. Ebbe bele tartoznak akár a mátrixok is, pontosabban a mátrixok modellezésére használt Matrix osztály. Egy későbbi projektben ha ilyen műveletek elvégzésére van szükség, akkor elegendő ezt a .jar fájlt hozzáadnunk a projekt könyvtáraihoz, és példányosítani a megfelelő objektumokat az aktuális problémának megfelelő formában. Ezzel a viszonylag egyszerű művelettel sikeresen megmenekültünk a mátrixokkal végezhető műveletek ismételt leimplementálásától, mivel készen felhasználhattuk őket. Ez persze nem csak Java nyelven oldható meg, de nem is ezt szerettem volna érzékeltetni, hanem azt, hogy a Java is képes erre. Ezen felül a Java nagy előnye mondjuk a C-vel kapcsolatban, hogy szinte minden terhet levesz a programozó válláról, mivel a Java osztályaiban már implementáltak nagyon sok olyan eszközt, amelyet a programozók leggyakrabban használnak, mint például a már C-ben is elérhető matematikai függvények, vagy a kollekciók megvalósítására illetve használatához szükséges eszközök. Ezért ha például összetett adatszerkezeteket szeretnénk használni, akkor nem kell bonyolult mutatózással és memóriafoglalási gondokkal foglalkoznunk, csak létrehozni a megfelelő objektumot, amely már valamely készen szállított osztály egy példánya lesz és az objektum metódusaival elvégezni a munka maradék részét. Például így nincs szükségünk rendező algoritmusok készítésére, mert a kollekciók rendezhetőek, csak azt kell megmondanunk, hogy mikor nagyobb vagy kisebb egy objektum a

másiknál, ha egy osztálynak a példányairól van szó. Az ilyen eszközöknek köszönhetően a rutinszerű feladatok helyett, amelyeket már minden programozó biztosan megoldott több alkalommal is, az üzleti haszonnal járó logika megalkotására fókuszálhatnak a fejlesztők. Meg kell viszont jegyezni azt is, hogy mint mindig az előnyök mellett beszélhetünk hátrányokról is. Ilyennek mondható az a tény is, hogy a Java programok futási ideje jóval nagyobb, mint az etalonként tekintett C programoké, amelyek ugyanazt a feladatot látják el. Ennek az is lehet az oka, hogy C-ben kicsit nagyobb befolyással rendelkezünk a memóriakezelésre, mivel ott mi dönthetjük el, hogy mikor foglaljuk le a memóriát és mikor szabadítjuk fel, míg ezzel szemben a Java szemétgyűjtögetős megoldásának köszönhetően nem okozhatunk ugyan olyan hibát, hogy nem szabadítjuk fel a területet a program futásának végeztével, de cserébe nem tudjuk azt sem megmondani, hogy a szemétgyűjtögetést nem most kell elkezdeni, hanem a most futó nagyon erőforrás-igényes programrész lefuttatása után. A másik ok, amelyik a lassúságot okozza, az a JVM, mivel a C programokat egy adott platformra fordítjuk, annak az összes előnyével és hátrányával, míg a Java bájtkódja csak egy köztes állapot, amelyet még értelmezni kell. Ennek köszönheti a Java a sebességcsökkenés egy részét is és a platformfüggetlenséget is. Ezen felül hátrány lehet az is, hogy az operációs rendszerekre külön fel kell telepíteni egyes esetekben a Java Virtuális Gépet, mert például a Microsoft Windows rendszerekre nincs telepítve alapból Sun JVM, még a Unix/Linux rendszerek egy jelentős részére sem. Igaz ugyan, hogy a Unixokon cserében van egy nem a Sun által fejlesztett, nyílt forráskódú változat, de az nem minden esetben szolgáltatja tökéletesen ugyanazt az eredményt, mint a Sun fejlesztésű társa. Itt jegyezném meg azt is, hogy a két teljesen mások által fejlesztett JVM között is lehetnek futási sebességben jelentkező különbségek.
(http://java.sun.com/new2java/programming/intro/ [2009.04.29. 22:14])

3.9 JUnit

A JUnit egy teszt keretrendszer, amelynek segítségével a szoftverfejlesztők által vétett hibák jelentős részét kényelmesen fel lehet deríteni. Manapság sajnos ha egy kezdő fejlesztő a tesztelésre gondol, akkor az negatív fogalomként hat, afféle felesleges időpazarlásnak tűnik számára. Ennek köszönhetően nem is szeretnek tesztelni, csak bíznak abban, hogy az a pár kattintás, amit a felületen elvégeztek, segít majd felderíteni azt a pár esetleges hibát, aminek a meglétét a programban feltételezik. Ez sajnos komoly probléma forrása lehet, mivel a felületen végzett tesztek nem derítenek ki minden hibát, amelyek a folyamatok mélyén fordulnak elő, illetve ha a fejlesztő végzi a tesztelést, akkor ő akaratlanul is a helyes működés felé fog tartani, mivel el sem tudja képzelni, hogy milyen igénybevételnek lesz kitéve a szoftver a felhasználók munkaállomásain. Ez tulajdonképpen azt jelenti, hogy a fejlesztő a hibák egy jelentős részét ilyen módon képtelen felderíteni, mert nem tudja, hogy mire kell számítania a szoftvernek, viszont cserébe pontosan tudja, hogy milyen gondolatmenet mentén épül fel a szoftver, amelyet tesztelni kell. Ezt elkerülendő célszerű pár alapvető gondolatot megfogalmazni a teszteléssel kapcsolatban:

- A komponenseket összeépítés előtt is tesztelni kell. Ha a termék nagyon bonyolult, akár több lépcsőre is szükség lehet (egy összeépített rendszer is esetleg csak egy komponens egy nagyobban)
- Szükség van egy külön tesztcsapatra, mert a fejlesztők egy csomó hibát nem látnak
- Alacsony szintű komponensek tesztelését csak a fejlesztők végezhetik, mert ehhez ismerniük kell a komponensek belső szerkezetét
- A funkcionális tesztelés - működik-e a termék? - csak egy része a tesztelők munkájának. Terhelési, stabilitási, használhatósági, stb. tesztekre is szükség lehet. (http://www.javagrund.hu/javasite/dokument/junit/index.html [2009.04.20. 18:14])

A JUnit keretrendszer ezekhez nyújt segítséget. Nem minden automatizálható vele még ma sem, de segítségével sokkal kényelmesebben végezhetünk például modulteszteket, mint nélküle. A legegyszerűbb tesztelési mód szerintem, ha minden osztályhoz létrehozunk egy JUnit tesztet, amelyben az osztály valamennyi metódusának tesztelhetjük a működését. Ez azért lehet jó, mert egy-egy metódusról még viszonylag könnyen meg lehet mondani, hogy milyen elvárásokat támasztunk vele szemben, azaz milyen bemenetre pontosan milyen

kimenetet kell generálnia, milyen kivételek váltódhatnak ki a működése során, illetve ezeknek pontosan mikor szabad kiváltódnia és mikor nem. Ehhez jelentős támogatottságot kapunk a JUnit-tól. Egyes integrált fejlesztői környezetekkel a tesztek megírásának a terhe sem hárul teljes egészében a fejlesztőre, mivel például a NetBeans IDE használata során pár kattintással sikeresen legenerálhatunk egy a teljes osztály minden metódusát tesztelő tesztgyűjteményt, amelyet nekünk már csak személyre kell szabnunk saját tesztesetekkel. Ez persze nagyobb projekteknél nagyon nehéz feladat is lehet, mivel a tesztesetek megtervezésekor a lehetséges hibák mindegyikét ki kellene tudnunk zárni ideális esetben, és ez nagy rendszereknél nagyon sok munkát jelent. Mindemellett ha rászánjuk az időt a tesztek megtervezésére, illetve végrehajtjuk azokat, akkor a felfedezett hibák kijavítását követően a rendszer egy magasabb minőségi szintjét állítjuk elő az előzővel szemben.

A JUnit modulteszt-keretrendszer főbb funkció a következők:

- Tesztek automatikus futtatása egyben vagy részenként
- Teszteredmény-áttekintés és kijelzés
- Hierarchikus tesztstruktúra-támogatás
- Tesztvégrehajtás többféle felületről

(http://www.javagrund.hu/javasite/dokument/junit/index.html [2009.04.20. 18:14])

Sajnálatos módon a JUnit sem tökéletes, mint ahogy semmi sem az. Emiatt meg kell említenem azt is, hogy mi az, amire nem képes a JUnit:

- Tesztek tervezése
- Automatikus tesztprogram generálás
- Lefedettség és teljesítménymérés

Ezeket tehát továbbra is a fejlesztő csapat valamely tagjainak kell végeznie.
(http://www.javagrund.hu/javasite/dokument/junit/index.html [2009.04.20. 18:14])
(http://junit.sourceforge.net/doc/faq/faq.htm#overview_1 [2009.04.29. 22:37])

3.10 i18n és L10n

Az i18n jelölés az angol internationalization szóból ered, pontosabban a kettő jelentését tekintve egy és ugyanaz. A háttérben az áll, hogy az internationalization szóban a szó eleji 'i' és a szó végi 'n' között pontosan 18 betű áll. Az i18n magyarul nemzetköziesítést jelent. Azért fontos ez a technológia, mert mint azt mindenki tudja, a világon sajnos több nyelv létezik, és nem mindenki beszéli mindegyiket, tehát a programok ha csak egy-egy nyelven léteznének, akkor nem mindenki tudná használni pusztán nyelvi akadályok miatt. Ez a kereskedelmi szoftverek esetében piacvesztést jelent, ami valószínűleg nem célja egyik cégnek sem, mivel nagyobb piacon többen vehetik meg a szoftvert. A nemzetköziesített szoftverek tehát előnyben vannak, mivel több nyelven jelennek megy egyszerre, így a felhasználók egy szélesebb csoportja lesz képes egyáltalán megérteni a program által használt nyelvet. Igen elterjedtek az olyan módszerek, ahol úgy nevezett nyelvfájlokban tárolják el a felhasználói felületen megjelenő tartalmak adott nyelvű szövegét, míg a program kódjába csak ezeknek valamilyen azonosítója van beillesztve, amely alapján a program be tudja azonosítani az oda illő szöveget az adott nyelvfájlon belül. Ez azért jó, mert a program kódjának megváltoztatása nélkül könnyedén módosíthatóak a felületen található szövegek mondjuk ha valamilyen hibát fedeztünk fel az egyik feliratban, vagy csak később jobb kifejezést találtunk valamire, mint az eredetileg beillesztett. Ezzel a programozók válláról jelentős terhet vehetünk le. A másik előny, hogy a nyelvfájlok programozói ismeretek nélkül is könnyedén lefordíthatóak, mivel csak arra kell figyelnünk, hogy az egyes szövegeket azonosító kulcsok ne változzanak meg, egyébként pedig minden egészen nyugodtan lefordítható. Ezt a munkát tehát bárki végezheti, aki rendelkezik a megfelelő nyelvismerettel, tehát megint a programozók feladataiból sikerült valamennyit átruházni másokra. A nemzetköziesítés egy megfelelően megírt alkalmazás esetén történhet jóval az alkalmazás kiadása után is, mivel az alkalmazás kódja illetve funkcionalitása ugyanaz marad csak a telepítő csomag változik azzal, hogy egy-egy új nyelvfájl kerül be a csomagba.

A nemzetköziesítés mellett a másik hasonló fogalom az L10n, amely az i18n-hez hasonló rövidítés, amely az angol localization szónak a rövidebb alakja. A localization magyarul lokalizációt jelent, amelyet Windowson a területi beállítások megadásától ismerhetünk, míg Linuxon a lokál (locale) beállításától. A lokalizáció lényege, hogy a

kifejlesztett szoftver megjelenése ne különbözzön a felhasználó kultúrája által megszabottaktól. Ez például azt jelenti, hogy ha a szoftver támogatja a magyar területi beállításokat, akkor a magyar viszonyokban megszokottaknak megfelelően a pénznem mindenhol Ft kell legyen, míg a hosszmértékegységek méterben lesznek kiírva, illetve az időbeállítások 24 órásak lesznek, a hét pedig hétfőn fog kezdődni. Sok minden más is jellemez még természetesen egy kultúrát, mint például az írás iránya, vagy a nemek szerepe, a helyi színek, az ünnepek, a földrajzi példák, stb., de most ezekre nem térnék ki külön. A lokalizált szoftver tehát teljes mértékben úgy kell kinézzen, mint egy az adott területen fejlesztett szoftverek, amelyek nyilván a saját területi beállításaikat fogják figyelembe venni. Ez egy nagyon nehéz feladat, mivel a lokalizációt végző csoportnak tökéletesen tisztában kell lennie a lokalizáció által befolyásolt valamennyi tényező az adott területre vonatkozó értével. Ennek hiányában elképzelhető, hogy a félig megvalósult lokalizáció hatására még rosszabb lesz a program, mint ha bele sem kezdtünk volna az egész folyamatba.
(http://searchcio.techtarget.com/sDefinition/0,,sid182_gci212496,00.html [2009.04.16. 12:05])

4. Összefoglalás

A dolgozat egy XML alapú matematikai alkalmazás kifejlesztéséről szól. Olvashattunk benne a megvalósítás főbb mozzanatai mellett a felhasznált technológiákról illetve a kész szoftver bevezetését valamint evolúcióját segítő tényezőkről. A kifejlesztett alkalmazás képességei megfelelnek a vele szemben támasztott elvárásoknak, a tervezett feladatok megvalósultak. Az alkalmazás hasznosítását tekintve a lehetséges célközönség az egyetemhez köthető, legyen szó oktatókról vagy hallgatókról egyaránt. Ennek ellenére én a hallgatók általi hasznosíthatóságot valószínűbbnek tartom, hiszen az egyetem bizonyára rendelkezik professzionális matematikai szoftverekkel, amelyek az oktatók munkáját elősegíthetik, míg az nem lenne megvalósítható, illetve egyáltalán ésszerű sem, ha minden hallgató beszerezné a komolyabb feladatokhoz előállított alkalmazásokat, majd azokat nem használnák ki maximálisan, mivel ezek gyakran sokkal nagyobb tudással rendelkeznek, mint amelyre egy átlagos hallgatónak szüksége van. Ezért a kifejlesztett alkalmazás megfelelő alternatívát biztosíthat az alacsonyabb bonyolultságú feladatok elvégzéséhez.

A program végleges funkcionalitása az alábbiakban olvasható:

- Infix vagy prefix matematikai kifejezések értékének meghatározása tetszőlegesen választható, akár 50000 számjegyes pontossággal.
- A szöveg bevitelének támogatása szintaktikai elemzéssel és a bemeneti mezőben található szöveg színezésével.
- XML dokumentumok előállítása és feldolgozása.
- Az eredmény XML dokumentumba mentése. Az előállított dokumentum böngészővel megjeleníthető a mentéskor létrehozott stíluslap segítségével
- A szöveges kifejezések elmentése későbbi feldolgozás lehetőségének biztosításához.
- Egyváltozós függvények képének kirajzolása 500x500 pixeles méretben.
- A függvények képének .png kiterjesztésű képekbe való mentése
- Angol és magyar nyelvű nyelvfájlok mellékelése, illetve a nyelvek listájának a továbbiakban lehetséges bővítésének biztosítása.
- Több platform támogatása: Windows és Unix/Linux rendszereken is használható az alkalmazás.
- Grafikus felhasználói felület.

A program továbbfejlesztésére is lehetőség van. Ezeket a lehetőségeket szeretném áttekinteni a következő szakaszban.

Az XML bemenet és kimenet lehetővé teszi, hogy a kifejezések kiértékelését végző motor egy webszolgáltatás működtetéséhez biztosítsa a számítások eredményét. Ehhez csupán a bejövő üzenetek feldolgoztatásával előállított válaszüzenetek visszaküldésének megoldására van szükség. Természetesen szükséges lehet biztonsági eszközök beépítésére is, vagy megfelelő naplózás megvalósítására, hogy a szolgáltatás igénybevétele szabályozható és megbízható legyen.

Könnyedén megoldható lenne továbbá egy parancssoros felhasználói felület megvalósítása is, hogy az alkalmazás akár olyan környezetekben is használható legyen, ahol nem lehetséges a grafikus felület miatt a futtatás jelen pillanatban. Ennek köszönhetően akár Unix szerverek SSH hozzáféréssel is funkcionálhatnának a számításigényes feladatok kiszolgálóiként.

Lehetőség van egy hálózati modul hozzáadására is, mivel a Java ehhez nagy segítséget nyújt. Ennek köszönhetően a nagyobb feladatok felbonthatóak lennének a kliens által annyi kisebb részre, ahány szervert talál a hálózaton, és a szerverek így párhuzamosan dolgozhatnak a feladat megoldásán. Ugyanezen technológia segíthetné a szoftver többmagos processzorokra történő optimalizálását is.

Hasznos lehet még a támogatott nyelvek pillanatnyilag eléggé rövid listájának bővítése, illetve a más platformok támogatásának megvalósítása is. Ezzel bővülhetne a felhasználók köre, illetve nőhetne a már meglévő felhasználók elégedettsége.

Megvalósítható lenne továbbá a postfix kifejezések értelmezése illetve kiértékelése is, illetve egy alkalmas precedencia-táblázat bevezetésével lehetőség nyílna a nem teljesen zárójelezett infix kifejezések feldolgozására is.

5. Irodalomjegyzék

- Dirk Louis-Peter Müller: Java 5 Belépés a programozás világába
  Panem Könyvkiadó 2006
  ISBN 963 545 4546
- Stoyan Gisbert: Numerikus matematika mérnököknek és programozóknak
  Typotex Elektronikus Kiadó 2007
  ISBN 9789639664418
- A JUnit teszt keretrendszer
  http://www.javagrund.hu/javasite/dokument/junit/index.html
  2009.04.20. 18:14
- Apfloat Java
  http://apfloat.org/apfloat_java/
  2009.04.27. 11:26
- Determinant
  http://mathworld.wolfram.com/Determinant.html
  2008.10.07. 9:08
- Gaussian elimination
  http://mathworld.wolfram.com/GaussianElimination.html
  2008.10.07. 11:12
- Greatest Common Divisor
  http://mathworld.wolfram.com/GreatestCommonDivisor.html
  2008.09.13. 9:23
- JUnit FAQ
  http://junit.sourceforge.net/doc/faq/faq.htm#overview_1
  2009.04.29. 22:37
- Least Common Multiple
  http://mathworld.wolfram.com/LeastCommonMultiple.html
  2008.09.13. 13:50

- New to Java
  http://java.sun.com/new2java/programming/intro/
  2009.04.29. 22:14
- SAX Quickstart
  http://www.saxproject.org/quickstart.html
  2009.04.29. 21:12
- The Document Object Model
  http://www.w3.org/DOM/
  2009.04.29. 20:05
- What is localization?
  http://searchcio.techtarget.com/sDefinition/0,,sid182_gci212496,00.html
  2009.04.16. 12:05
- Window To Viewport Transformation
  http://www.siggraph.org/education/materials/HyperGraph/viewing/view2d/pwint.htm
  2008.10.03. 10:32
- XML 10 pontban
  http://www.w3c.hu/forditasok/XML_10_pontban.html
  2009.04.10. 19:01
- XML in 10 points
  http://www.w3.org/XML/1999/XML-in-10-points.html
  2009.04.27. 11:00

6. Függelék

6.1 A kifejlesztett alkalmazás magyar nyelvű felhasználói kézikönyve

XML Calculator Felhasználói kézikönyv

Tartalomjegyzék:

1. Bevezetés

Az XML Calculator egy matematikai feladatok elvégzésére tervezett alkalmazás. A program célja, hogy képes legyen teljesen zárójelezett infix, illetve prefix kifejezések kiértékelésére. Használható mátrixokkal való műveletek elvégzésére és függvényrajzolásra is. Hasznos alkalmazás lehet egyetemen, főiskolán matematikai tanulmányokat folytató hallgatók számára.

2. Rendszerkövetelmények

250 MHz-es AMD vagy Intel processzor

128 MB rendszermemória

10 MB szabad terület a merevlemezen

Videokártya

Minimum 1024x768-as felbontású képernyő

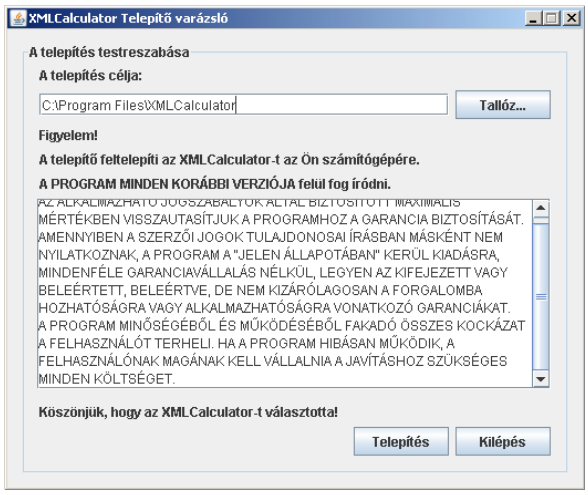
Java Runtime Environment 6

Microsoft Windows XP/Vista/2003/2008 vagy Linux

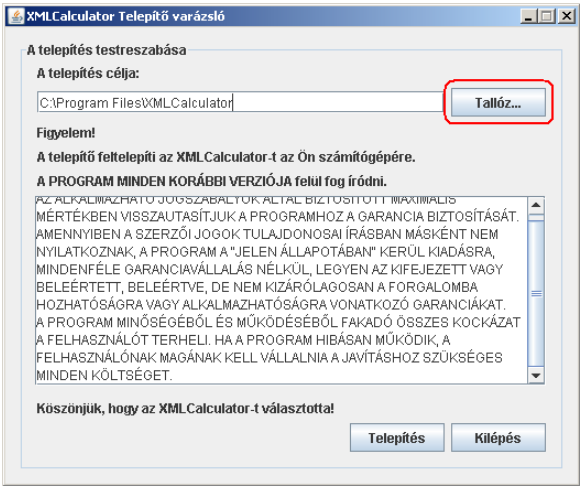
3. Telepítési útmutató

a) Ha Ön Microsoft Windows operációs rendszert használ

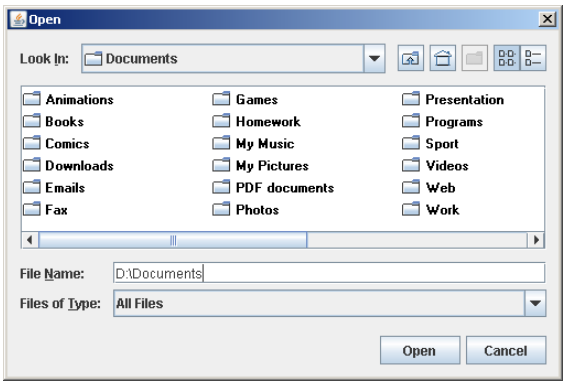
1. Kattintson duplán az ezen dokumentummal egy könyvtárban található 'Install_XMLCalculator.jar' nevű alkalmazásra! Ekkor meg kell jelennie az alábbi ábrán látható ablaknak.



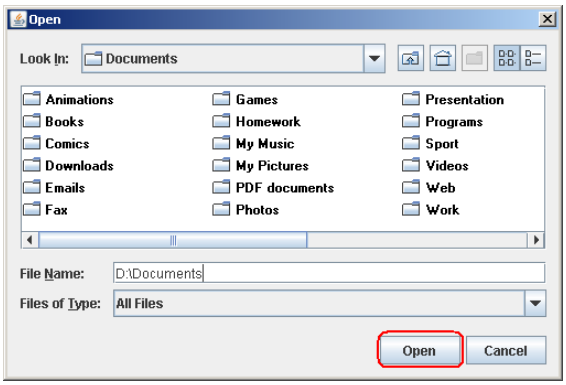
2. Ekkor kiválaszthatja az Ön által legjobbnak vélt telepítési célt, ahová a telepítő a fájlokat be fogja másolni. Ezt a 'Tallóz...' gomb megnyomásával teheti meg, amelynek a helye az alábbi ábrán látható:



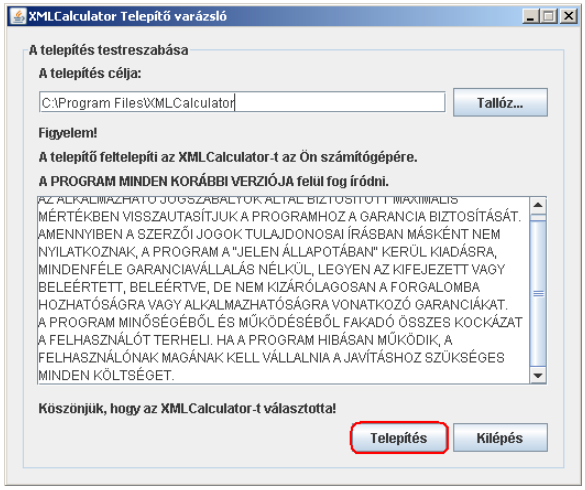
Erre a gombra kattintva egy, a következő képen láthatóhoz hasonló ablak fog megjelenni.



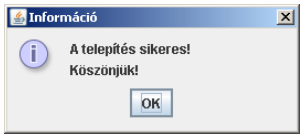
A megjelenő ablakban válassza ki a kívánt célkönyvtárat! Figyelem! A telepítő közvetlenül a kiválasztott könyvtárba fogja másolni a fájlokat. Ha szeretne új könyvtárat létrehozni, azt Önnek kell megtennie, vagy manuálisan át kell írnia a már kiválasztott könyvtár elérési útját az 'Open' gomb megnyomását követően a Tallózás gombtól balra található mezőben.
Ha sikerült kiválasztania a könyvtárat, nyomja meg az 'Open' gombot!



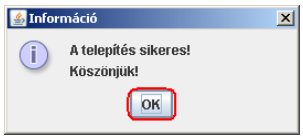
3. Ha a telepítés célja mezőben Önnek megfelelő könyvtár elérési útvonala van beírva, akkor nyomja meg a 'Telepít' gombot!



4. Ekkor, ha sikeres volt a telepítési eljárás, akkor arról egy információs üzenetet fog kapni, ami így néz ki:

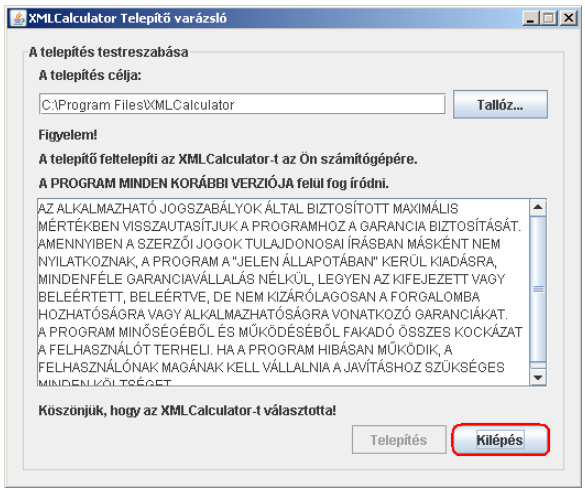


Ezen a párbeszédpanelen kattintson az 'OK' gombra!



5. Ezután a 'Kilépés' gomb megnyomásával zárja be a varázslót! Ezzel sikeresen befejezte a program telepítését.

   Köszönjük, hogy ezt az alkalmazást választotta!



b) Ha Ön Linux operációs rendszert használ

1. Nyisson meg egy parancsértelmezőt, majd navigáljon abba a könyvtárba, amelyben ezt a dokumentumot találta.
2. Adja ki a következő parancsot:

   ```
   java -jar Install_XMLCalculator.jar
   ```

3. Ezután kövesse a Microsoft Windowsra leírt lépéseket a 2. lépéstől az utolsóig!

4. A program eltávolítása

Törölje a telepítéskor megadott útvonalon található könyvtár tartalmát!

Emellett Windowson törölheti a program beállításait a %APPDATA% környezeti változó által mutatott könyvtárban található XMLCalculator mappa törlésével.

Linuxon a home könyvtárából a .XMLCalculator mappa törlésével teheti meg ugyanezt.

5. A program használata

a) Infix kifejezések

A program csak teljesen zárójelezett infix kifejezéseket, vagy prefix kifejezéseket tud használni. Ez a szakasz a teljesen zárójelezett infix kifejezések bemutatására szolgál.

Az infix kifejezéseket az jellemzi, hogy a kétoperandusú operátorok infix leírásakor az operátor a két operandus között helyezkedik el.

pl.: 3 + 2

A fenti példában a 3 és a 2 operandusok között található a + operátor.

Az egyoperandusú, illetve a programban található speciális operátorok esetében más a helyzet, mivel itt nem beszélhetünk biztosan két operandusról. Ezeknél az operátor megelőzi az összes operandusát, illetve a speciális operátoroknál egy külön jelzésre van szükség, hogy tudjuk, mennyi operandusa van jelen esetben az operátornak. Erről a későbbiekben lesz szó.

Az egyoperandusú operátorokat tehát egyszerűen úgy írjuk le, hogy előre az operátort, majd azt követően az operandust.

pl.: sin 0

Ebben a példában a sinus operátor megelőzi a nulla operandusát.

A speciális operátorok esetében az operátor megelőzi az operandusait, de mivel az operandusok számossága nem ismert számunkra, kénytelenek vagyunk az operandusokat egy {} zárójelpár közé írni egymástól szóközzel elválasztva.

pl.: min{1 2 3 4 5 6 7 8}

A példában látható sor egy minimum operátort mutat be, amelynek az operandusai a számok egytől nyolcig.

A teljesen zárójelezésre azért van szükség, mert egyébként az operátorok kiértékelésének sorrendje nem lenne egyértelmű, ami bozonyos esetekben gondot okozhatna:

*pl.: 3 + 5 * 2 = ?* *3 + (5 * 2) ≠ (3+5) * 2*

A fenti példában látható eset például egy ilyen, amikor nem mindegy, hogy *3+(5*2)* vagy *(3+5)*2* módon értékeljük-e ki a *3+5*2* kifejezést. Ezért használjuk az egyértelmű, teljesen zárójelezett *(((3)+(5))*(2))* vagy *((3)+((5)*(2)))* alakok

valamelyikét.

A teljesen zárójelezett kifejezések felépítésére a következő szabályok érvényesek:

- Mindig van pontosan egy külső zárójelpár.
- Egyetlen szám *(szám)* formában írandó.
- Egy egyoperandusú operátor *(operátor(operandus))* alakban írandó, ahol az operandus egy tetszőleges az operátor értelmezési tartományának megfelelő értékkel bíró kifejezés lehet.

  pl.: (sin(0))
- Egy kétoperandusú operátor *((operandus1)operátor(operandus2))* alakban írandó, ahol az operandus1 és az operandus2 az operátor értelmezési tartományából származó értékkel rendelkező kifejezések.

  pl.: ((3)+(2))
- Egy speciális operátor *(operátor{(op.1) (op.2) ... (op.n)})* alakban írandó, ahol az op.1 op.2 … op.n operandusok (n darab) az operátor értelmezési tartományából származó értékkel rendelkező kifejezések.

  pl.: *(min{(1) (2) (3) (4) (5) (6) (7) (8)})*

Mátrixokkal való műveletek végzésekor is érvényesek a fenti szabályok, csak a mátrixok határait kell jeleznünk a [és a] jelekkel, amelyek a mátrix kezdetét illetve végét jelzik. A mátrixokat soronként kifejtve kell megadni a programnak, ami azt jelenti, hogy a [jel után kezdődik az első sor elemeinek a felsorolása, majd ha vannak még további sorok, akkor egy | jel következik, amely a sorok elválasztására hivatott, és a következő sor elemeivel folytatódhat a mátrix leírása. Az utolsó sor leírása után] jellel jelezzük a mátrix végét. Természetesen a külső zárójelekről nem szabad megfeledkeznünk.

pl.: *([(1) (2) (3) | (4) (5) (6)])*

A példában szereplő mátrix a következő mátrixnak a teljesen zárójelezett infix alakja:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

b) *Prefix kifejezések*

A prefix írásmód esetén az egyoperandusú illetve speciális operátorok írásmódja megegyezik az infix írásmódnál leírtakkal, azaz az operátor megelőzi az összes operandusát. Az eltérést a kétoperandusú operátoroknál tapasztalhatjuk, mivel az ilyen operátorok prefix alakjában az operátort követi a két operandus.

pl.: + 3 2

Emellett meg kell említenünk, hogy ebben az esetben a kerek zárójelek használata teljesen felesleges, mivel a prefix operátorok egyértelműek, azaz nem lehet „félreérteni” a kifejezésben a kiértékelés kívánt sorrendjét. Ennek köszönhetően a program nem engedélyezi a kerek zárójelek használatát prefix kifejezések esetén.

c) *Függvényrajzolás*

A program képes a beírt kifejezésekből függvények rajzolására. Amennyiben szeretné használni ezt a funkciót, akkor szerkessze meg a kívánt kifejezést a 'Kifejezés-szerkesztő' fület használva, vagy töltsön be egy korábban már elmentett dokumentumot a 'Beállítások' fülön, és kattintson a 'Rajzold!' gombra. Fontos, hogy amennyiben most szerkeszti meg a kifejezést, akkor a 'Kifejezés-szerkesztő' fülön, ha pedig csak betöltött egy korábbi kifejezést, akkor a 'Beállítások' fülön nyomja meg a 'Rajzold!' gombot.

A kirajzolható kifejezésekre léteznek megkötések, amelyeket ebben a bekezdésben ismertetünk. A kifejezés nem tartalmazhat mátrixokat sehol, amennyiben ki szeretnénk rajzolni a képét. A kirajzolandó kifejezés egy egyváltozós függvény lehet. Ehhez a változó értéket a VAR kulcsszó operandusként történő begépelésével, vagy a kurzorral a kívánt operandus helyén állva a változó gomb megnyomásával teheti meg a 'Kifejezés-szerkesztő' fülön. Ez azt eredményezi, hogy a megadott vízszintes intervallumot bejárja a VAR értéke a megadott mérési pontok számának megfelelő sűrűségben, és a program görbe függőleges koordinátáját az aktuális mérési ponthoz tartozó vízszintes koordinátaértékét a kifejezésbe helyettesítve számítja ki.

A képpel kapcsolatos beállításokat a 'Függvény beállítások' fülön találja, ahol megadhatja a kirajzolandó intervallum kezdő illetve végértékeit vízszintes illetve függőleges irányban, illetve a kapott kép minőségét befolyásoló tényezőket. A

kirajzolandó területet méretének beállítását a felső 2x3 db csúszka segítségével teheti meg. A rajzolás minőségét több tényező is befolyásolja. Ezek egyike maga a kifejezés, amelyet Ön ki szeretne rajzolni. Bizonyos esetekben a rajz eltérhet a függvény valós képétől, amely technikai okokból van így. Különösen problémás lehet a tangens illetve egyéb meredek függvények rajzolása. A második tényező a mérési pontok száma. Ez azt adja, meg, hogy a kapott intervallumon belül hány helyen vegyen mintát a program a függvény értékéből. Ez a szám minél nagyobb, annál pontosabb képet rajzol a program a függvényről, viszont sajnálatos módon annál lassabb a függvény kirajzolása. Ez az idő akár több perc is lehet. A harmadik, és egyben utolsó tényező a pontosság, ami a számítás pontosságát jelenti. Ez a szám azt mutatja, hogy hány számjegy pontos a szám elejétől számítva, például az 102.5453 szám pontossága 7. Amennyiben ennél kisebb érték van megadva pontosságnak, akkor torzul a szám értéke. Ha a pontosság nagy, akkor is javulhat a rajz minősége, de jelentősen lassul a rajzolási folyamat. Ezen a fülön kell kiválasztani azt is, hogy a függvény és a koordináta-tengelyek metszéspontjai ki legyenek-e számítva. Ezek lehetnek például zérushelyek is, vagy a függőleges tengely és a függvény metszéspontja is.

d) *Kifejezések kiértékelése*

Kifejezések kiértékeléséhez szerkessze meg a kívánt kifejezést a 'Kifejezés-szerkesztő' fület használva, vagy töltsön be egy korábban már elmentett dokumentumot a 'Beállítások' fülön, és kattintson a 'Számold!' gombra. Fontos, hogy amennyiben most szerkeszti meg a kifejezést, akkor a 'Kifejezés-szerkesztő' fülön, ha pedig csak betöltött egy korábbi kifejezést, akkor a 'Beállítások' fülön nyomja meg a 'Számold!' gombot. Ebben az esetben nem használhat változókat a kifejezésben, mert az nem értelmezett.

e) *Műveletek fájlokkal*

1. Dokumentumok készítése:

   Ha új dokumentumot szeretne készíteni, és azt elmenteni, akkor szerkessze meg a kívánt kifejezést a 'Kifejezés-szerkesztő' fület használva, majd a 'Beállítások' fülön válassza ki a 'Mentsd el a kifejezést feldolgozás előtt' panelben 'A kifejezés fájl helye...' feliratú mező melletti Tallóz... gombot, és

válassza ki a kimeneti xml fájl helyét, illetve nevét. Ellenőrizze le, hogy a mező felett be van-e jelölve a jelölőnégyzet, illetve, hogy a mezőben megjelent-e az Ön által kívánt fájlnév. Ezután a 'Kifejezés-szerkesztő' fülön kattintson a 'Mentés XML-be' gombra. Ezzel sikeresen elmentette a kifejezését feldolgozás nélkül. Ez persze csak akkor igaz, ha nincs hiba a kifejezésben, mivel az előfeldolgozás ekkor is megtörténik, és ennek során már a kifejezés helyességének ellenőrzés is. Ha nem a 'Mentés XML-be' gombra kattint, hanem kirajzoltatja, vagy kiszámoltatja a kifejezés képét, vagy értékét, akkor is mentésre kerül a kifejezése. Fontos, hogy nem egy, hanem két fájl jön ekkor létre. Az egyik a kifejezést tartalmazó .xml, a másik pedig, a helyességének leellenőrzését segítő .xsd fájl. Amennyiben mozgatja a .xml fájlt, mozgassa vele a .xsd-t is, mert egyébként nem fogja tudni használni a kifejezést.

Új dokumentumot hozhat létre akkor is, ha .xml-be szeretné menteni a számítás eredményét. Ekkor az előbbi alatt az 'Az eredmény mátrix vagy szám mentése' feliratú panelen kell az előzőekhez hasonlóan kiválasztania a kimeneti fájlt, majd nem a 'Mentés XML-be', hanem a 'Számold!' gombra kattintani a 'Kifejezés-szerkesztő' fülön. Ebben az esetben még egy fájl jön létre a mentéssel, amelynek .xsl a kiterjesztése. Ez egy stíluslap, amely arra szolgál, hogy a .xml fájlt böngészőben megjeleníthesse.

A dokumentumok mellett arra is van lehetősége, hogy elmentse a kirajzolt képét a függvényeknek. Ezt a 'Függvény beállítások' fül alján található panelben teheti meg a kép kitallózásával az eddigiekben már bemutatott módokhoz hasonlóan. Az eredmény egy .png kép lesz.

2. Dokumentumok felhasználása:

   Korábban mentett dokumentumait a 'Beállítások' fülön nyithatja meg az 'Egy xml kifejezés dokumentum megnyitása' panelben található 'Tallóz...' gombbal. Ezután a kifejezés dokumentum elérési útvonala megjelenik a gomb melletti mezőben. Ha a mező alatti 'Számold!' illetve 'Rajzold!' gombok valamelyikére kattint, akkor a program megpróbálja kiírni vagy kirajzolni a kifejezésnek megfelelően az eredményt. A függvények rajzolásakor megadható beállítások, azaz az intervallum mérete, és a rajz minősége ekkor is módosítható.

6. Az operátorok leírásai

Abszolút érték:

Szöveges alak: *abs* **Xml címkék:** *abs, absolutevalue*

Típus: egyoperandusú

Leírás: Számokra alkalmazható operátor, amely kiszámítja az operandus abszolút értékét, azaz, ah az operandus<0, akkor negálja, egyébként pedig nem változtat az operandus értékén.

Aritmetikai-geometriai közép:

Szöveges alak: *agm* **Xml címkék:** *agm, arithmeticgeometricmean*

Típus: kétoperandusú

Leírás: Kiszámítja a két operandus aritmetikai-geometriai közepét.

Binomiális együttható:

Szöveges alak: *bic* **Xml címkék:** *bic, binomialcoefficient*

Típus: kétoperandusú

Leírás: A két természetes szám típusú operandusból binomiális együtthatót számít, azaz ha az első operandus n, a második pedig k, akkor n alatt a k értékét adja meg, ha n nem kisebb, mint k.

Cosinus-hyperbolicus:

Szöveges alak: *cosh* **Xml címkék:** *cosh, hyperboliccosine*

Típus: egyoperandusú

Leírás: A valós szám típusú operandus cosinus-hyperbolicusát adja meg

Cosinus:

Szöveges alak: *cos* **Xml címkék:** *cos, cosine*

Típus: egyoperandusú

Leírás: A valós szám típusú operandus cosinusát adja meg.

Determináns:

Szöveges alak: *det* **Xml címkék:** *det, determinant*

Típus: egyoperandusú

Leírás: A kvadratikus mátrix típusú operandus determinánsát számítja ki. Ha a kapott mátrix nem kvadratikus, akkor nem tudja elvégezni a műveletet. Egy *1x1*-es mátrixnál, vagy számnál az operandust valós számmá alakítva adja meg.

Exponens:

Szöveges alak: *exp* **Xml címkék:** *exp, exponent*

Típus: egyoperandusú

Leírás: A valós szám típusú operandusnak megfelelő hatványra emeli az e számot, azaz ha az operandus x, akkor e^x -et számítja ki.

Faktoriális:

Szöveges alak: *!* **Xml címkék:** *fact, factorial*

Típus: egyoperandusú

Leírás: A természetes szám típusú operandus faktoriálisának értékét számítja ki.

Gyökvonás:

Szöveges alak: *root* **Xml címke:** *root*

Típus: kétoperandusú

Leírás: A valós szám típusú első operandusból és az egész típusú második operandusból, ha az első operandus n, a második pedig k, akkor $\sqrt[k]{n}$ értékét számítja ki. Természetesen ha k páros és n negatív, akkor nem végezhető el a művelet, illetve amennyiben n nulla, akkor sem.

Hatványozás:

Szöveges alak: ^ **Xml címkék:** *pow, raisetopower*

Típus: kétoperandusú

Leírás: A két valós szám típusú operandusból, amennyiben az első operandus n, a második pedig k, akkor n^k értékét számítja ki, feltéve, hogy n és k egyszerre nem nulla, illetve ha az első operandus kisebb, mint nulla, akkor sem végződik el a művelet.

Invertálás:

Szöveges alak: *inv* **Xml címkék:** *inv, invert*

Típus: egyoperandusú

Leírás: A mátrix típusú operandus inverzét számítja ki, ha a mátrix kvadratikus, illetve a determinánsa nem nulla. Ha az operandus valós szám, akkor a számot *1x1*-es mátrixként értelmezve, az inverz a szám reciproka lesz, így az operátor azt adja eredményül.

Inverz cosinus-hyperbolicus:

Szöveges alak: *acosh* **Xml címkék:** *acosh, inversehyperboliccosine*

Típus: egyoperandusú

Leírás: A valós szám típusú operandusból a cosinus-hyperbolicus függvény inverzével meghatározott értéket adja meg. Tehát ha az operandus x, akkor $ch^{-1}(x)$ -et adja eredményül. Amennyiben x kisebb, mint 1, a függvény nem értelmezett.

Inverz cosinus:

Szöveges alak: *acos* **Xml címkék:** *acos, inversecosine*

Típus: egyoperandusú

Leírás: A valós szám típusú operandusból a cosinus függvény inverzével meghatározott értéket adja meg. Tehát ha az operandus x, akkor $cos^{-1}(x)$ -et adja eredményül. Amennyiben x kisebb, mint -1 vagy nagyobb, mint 1, a függvény nem értelmezett.

Inverz sinus-hyperbolicus:

Szöveges alak: *asinh* **Xml címkék:** *asinh, inversehyperbolicsine*

Típus: egyoperandusú

Leírás: A valós szám típusú operandusból a sinus-hyperbolicus függvény inverzével meghatározott értéket adja meg. Tehát ha az operandus x, akkor $sh^{-1}(x)$ -et adja eredményül.

Inverz sinus:

Szöveges alak: *asin* **Xml címkék:** *asin, inversesine*

Típus: egyoperandusú

Leírás: A valós szám típusú operandusból a sinus függvény inverzével meghatározott értéket adja meg. Tehát ha az operandus x, akkor $sin^{-1}(x)$ -et adja eredményül. Amennyiben x kisebb, mint -1 vagy nagyobb, mint 1, a függvény nem értelmezett.

Inverz tangens-hyperbolicus:

Szöveges alak: *atanh* **Xml címkék:** *atanh, inversehyperbolictangent*

Típus: egyoperandusú

Leírás: A valós szám típusú operandusból a tangens-hyperbolicus függvény inverzével meghatározott értéket adja meg. Tehát ha az operandus x, akkor $th^{-1}(x)$ -et adja eredményül. Amennyiben $|x| \geq 1$, a függvény nem értelmezett.

Inverz tangens:

Szöveges alak: *atan* **Xml címkék:** *atan, inversetangent*

Típus: egyoperandusú

Leírás: A valós szám típusú operandusból a tangens függvény inverzével meghatározott értéket adja meg. Tehát ha az operandus x, akkor $tg^{-1}(x)$ -et adja eredményül.

Kivonás:

Szöveges alak: - **Xml címkék:** *sub, subtraction*

Típus: kétoperandusú

Leírás: A két operandus különbségét adja meg, amennyiben az operandusok dimenziója megfelelő. Számítható vele két szám különbsége, vagy két tetszőleges *nxm*-es mátrix különbsége is.

Köbgyökvonás:

Szöveges alak: *cbrt* **Xml címkék:** *cbrt, cuberoot*

Típus: egyoperandusú

Leírás: A valós szám típusú operandus köbgyökét számítja ki, azaz $\sqrt[3]{n}$ -et amennyiben az operandus n.

Legkisebb közös többszörös:

Szöveges alak: *lcm* **Xml címkék:** *lcm, leastcommonmultiple*

Típus: kétoperandusú

Leírás: A két egész szám típusú operandus legkisebb közös többszörösét számítja ki.

Legnagyobb közös osztó:

Szöveges alak: *gcd* **Xml címkék:** gcd, greatestcommondivisor

Típus: kétoperandusú

Leírás: A két egész szám típusú operandus legnagyobb közös osztóját számítja ki.

Logaritmus:

Szöveges alak: *log* **Xml címkék:** *log, logarithm*

Típus: egyoperandusú

Leírás: A pozitív valós szám típusú operandus természetes alapú logaritmusát számítja ki.

Maradék képzés:

Szöveges alak: *%* **Xml címkék:** *mod, modulo*

Típus: kétoperandusú

Leírás: A két valós operandus maradékos osztásakor keletkező maradékát számítja ki, feltéve, hogy az osztó nem nulla.

Maradék nélküli osztás:

Szöveges alak: */* **Xml címkék:** *div, divisionwithnoreminder*

Típus: kétoperandusú

Leírás: A két valós operandus hányadosát számítja ki, feltéve, hogy az osztó nem nulla.

Maradékos osztás:

Szöveges alak: */%* **Xml címkék:** *divr, divisionwithreminder*

Típus: kétoperandusú

Leírás: A két valós operandus maradékos osztásakor keletkező hányadosát számítja ki, feltéve, hogy az osztó nem nulla.

Maximum:

Szöveges alak: *max* **Xml címkék:** *max, maximum*

Típus: speciális

Leírás: A kapott valós szám típusú elemekből álló halmaz legnagyobb elemét keresi meg, és azt adja eredményül.

Minimum:

Szöveges alak: *min* **Xml címkék:** *min, minimum*

Típus: speciális

Leírás: A kapott valós szám típusú elemekből álló halmaz legkisebb elemét keresi meg, és azt adja eredményül.

Negálás:

Szöveges alak: *neg* **Xml címkék:** *neg, negate*

Típus: egyoperandusú

Leírás: Negálja a valós szám típusú operandust, azaz -1-gyel szorozza azt.

Négyzetgyökvonás:

Szöveges alak: *sqrt* **Xml címkék:** *sqrt, squareroot*

Típus: egyoperandusú

Leírás: A pozitív valós szám típusú operandus négyzetgyökét számítja ki, azaz $\sqrt[2]{n}$ -et amennyiben az operandus n.

Norma:

Szöveges alak: *abs* **Xml címke:** *norm*

Típus: egyoperandusú

Leírás: A kapott *nxm*-es mátrix végtelen normáját számítja ki, ha $n>1$ és $m>1$. Amennyiben $n=1$ és $m>1$ vagy $n>1$ és $m=1$, akkor a vektor 2 normáját számítja ki az operátor. Ha a mátrix *1x1*-es, akkor abszolút-értékként funkcionál.

Összeadás:

Szöveges alak: + **Xml címkék:** *add, addition*

Típus: kétoperandusú

Leírás: A két operandus összegét adja meg, amennyiben az operandusok dimenziója megfelelő. Számítható vele két szám összege, vagy két tetszőleges *nxm*-es mátrix összege is.

Produktum:

Szöveges alak: *prd* **Xml címkék:** *prd, product*

Típus: speciális

Leírás: A kapott valós szám típusú elemekből álló halmaz elemeinek a szorzatát adja eredményül.

Sinus-hyperbolicus:

Szöveges alak: *sinh* **Xml címkék:** *sinh, hyperbolicsine*

Típus: egyoperandusú

Leírás: A valós szám típusú operandusból a sinus-hyperbolicus függvénnyel meghatározott értéket adja meg. Tehát ha az operandus x, akkor *sh(x)* -et adja eredményül.

Sinus:

Szöveges alak: *sin* **Xml címkék:** *sin, sine*

Típus: egyoperandusú

Leírás: A valós szám típusú operandus sinusát adja meg.

Szorzás:

Szöveges alak: * **Xml címkék:** *mul, multiplication*

Típus: kétoperandusú

Leírás: Összetett operátor. Ha az operandusai valós számok, akkor a szorzatukat adja meg, egyébként használható mátrixok szorzására is a mátrix-szorzás szabályainak megfelelően, illetve vektorok skaláris vagy belső szorzatának kiszámítására is, ha mindkét operandus *1xn*-es vagy mindkét vektor *nx1*-es.

Szumma:

Szöveges alak: *sum* **Xml címke:** *sum*

Típus: speciális

Leírás: A kapott valós szám típusú elemekből álló halmaz elemeinek az összegét adja eredményül.

Tangens-hyperbolicus:

Szöveges alak: *tanh* **Xml címkék:** *tanh, hyperbolictangent*

Típus: egyoperandusú

Leírás: A valós szám típusú operandusból a tangens-hyperbolicus függvénnyel meghatározott értéket adja meg. Tehát ha az operandus x, akkor *th(x)* -et adja eredményül.

Tangens:

Szöveges alak: *tan* **Xml címkék:** *tan, tangent*

Típus: egyoperandusú

Leírás: A valós szám operandus tangensét adja meg, feltéve, hogy a tangens értelmezett az operandusra.

Transzponálás:

Szöveges alak: *tr* **Xml címkék:** *tr, transponate*

Típus: egyoperandusú

Leírás: A mátrix típusú operandus transzponáltját adja eredményül. Ha az operandus valós szám, akkor az operandus értékét adja eredményül.

7. Jogi közlemény

Az alkalmazás nem jöhetett volna létre a Mikko Tommila által kifejlesztett Apfloat nevű matematikai eszközgyűjtemény nélkül. A www.apfloat.org oldalon olvasható legfontosabb információk magyarra fordítva az alábbiakban olvashatóak:

„Az Apfloat egy nagy teljesítményű Tetszőleges pontosságú aritmetikai könyvtár. Több millió számjegyes pontossággal végezhetünk számításokat a segítségével. Ugyanolyan egyszerű használni, mint a Java BigDecimal vagy BigInteger osztályai, de sokkal jobban teljesít extrém pontosságú számokkal (több, mint egy pár száz számjegy). Emellett egy teljes csoportja elérhető a matematikai függvényeknek tetszőleges pontosságú számokhoz: az összes java.lang.Math-ben elérhető és továbbiak. Az Apfloat a GNU Lesser General Public License (2.1-es verzió vagy bármely későbbi) alatt érhető el, és NINCSEN GARANCIA rá.”

7. Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani Dr. Adamkó Attilának, a témavezetői feladatok elvégzéséért, mert az ő közreműködése nélkül ez a dolgozat nem jöhetett volna létre. Köszönöm továbbá az Informatikai Kar valamennyi oktatójának, hogy a munkájuknak köszönhetően fel tudtam készülni a dolgozat megírására. Meg kell még említenem, hogy köszönet illeti az Apfloat eszközök fejlesztőjét, Mikko Tommila-t, mert ha nem fejlesztette volna ki ezt a nagyszerű csomagot, akkor nem tudtam volna létrehozni ezt az alkalmazást. Szeretném megköszönni mindenki másnak is, aki támogatott a munkában, vagy bármilyen segítséget nyújtott a tanulmányaim során.