Ingyenes CAS bármely operációs rendszerre: a HartMath

Egy Javában íródott, GPL szerződés alá eső Számítógépes Algebra Rendszert szeretnék most bemutatni. Okos és gyors – még a mai grafikus számológépek mellett is tartalmaz meglepetéseket.

Ha még nem hallottál a Számítógépes Algebra Rendszerről (CAS), röviden elég annyit tudnod, hogy olyan számológép, amely számok mellett képleteket is kezel. Vannak változói, képes törteket gyökteleníteni, és még rengeteg feladatot megoldani, hogy a matematikai műveleteknél véletlenül se számoljuk el magunkat. Programozható is, lehet benne ciklusokat, elágazásokat készíteni. A ➲ http://freshmeat.net-en keresgélve két ígéretes fejlesztést is találtam. Az egyik a Yacas (Yet Another Computer Algebra System), amely a ➲ http://www.xs4all.nl/~apinkus/yacas.html címen érhető el, a másik pedig a HartMath ➲ http://www.hartmath.org. Ez utóbbival szerzett tapasztalataimat szeretném most megosztani veletek.

A telepítés

A most következő lépések az előfordított Java *.jar* állomány telepítését írják le. A program forráskódja természetesen nyílt, bárki lefordíthatja, én azonban maradtam a kész *.jar* fájlnál. Mindenekelőtt ha még nem rendelkeznél Java Virtuális Géppel (JVM), telepíts egyet. Javaslom a Sun-féle J2RE (Java 2 Runtime Environment) megvalósítást. Ez jelenleg az 1.3.1-es változatnál tart, eléggé megbízható, és viszonylag gyors is. A telepítőt a ➲ http://java.sun.com címről ingyenesen letölthetted. Ez egy *.bin* kiterjesztésű állomány, és ha kiadod a `bash j2re-verzio.bin` parancsot, a felhasználási szerződés elfogadása után szó nélkül kicsomagolja magát a pillanatnyi könyvtárba. Én ezt követően átmásoltam a */opt*-ba, és */usr/bin/java* néven készítettem egy közvetett hivatkozást a */opt/jre1.3.1_02/bin/java* állományra (amely szintén csak egy hivatkozás, de ez ebből a szempontból lényegtelen). Mivel a */usr/bin* benne van a `PATH` környezeti változóban, a Java már futtatható. Ellenőrzésképpen beírhatod a következőt:

```
$ java -version
```

Amire én ezt kaptam:

```
java version  1.3.1_02
Java(TM) 2 Runtime Environment,
➥Standard Edition (build 1.3.1_02-b02)
Java HotSpot(TM) Client VM (build
➥1.3.1_02-b02, mixed mode)
```

Ezekután itt az idő, hogy beszerezd a HartMathot. A fentebb említett címről letölthetsz egy Javában írt telepítőt. Ha megvan a *hartmathverzioinstall.jar* fájl, a java `-jar fajlnev.jar` paranccsal indítsd el egy terminálról. Ez azonban grafikus felületet igényel, tehát a legjobb, ha az X-et rendszergazdaként indítod (és utána gyorsan ki is lépsz). A telepítés innentől már nagyon egyszerű. Én a */opt/HartMath07pre19* könyvtárba telepítettem, és egyúttal ide tettem a rendszer pdf formátumú kézikönyvét. Ez nincs benne a telepítőcsomagban, ugyancsak a fenti címről külön tölthető le.

Ismerkedés a rendszerrel

Javáról lévén szó nem lepődtem meg azon, hogy az ablak két kattintás után nem pattan fel azonnal a képernyőre. Egy kicsit bosszantott, hogy az `xosview`-t figyelve a program az indítás során 17 MB fizikai memóriát evett meg, és ekkor még egy gombot sem nyomtam le. A kezelőfelület szépnek nem nevezhető, de könnyen átlátható és tanulható. A felső menüsorban a már megszokott *File* mellett, az *Edit* alatt érheted el a vágólappal kapcsolatos szolgáltatásokat: másolás, kivágás és beillesztés. A további két menüben a gyakran használt függvények és példák kaptak helyet.

A menüsor alatt található beviteli mezőben van lehetőség a képletek beírására és szerkesztésére. Ez alatt számológéphez hasonló elrendezésű gombok kaptak helyet. A számok, zárójelek, logikai és egyéb műveleti jelek mellett itt találhatod meg a 16-os (0x), a 8-as (0o), és a 2-es (0b) számrendszerhez tartozó előtagokat (prefix). Ezek mellett jobb oldalt található egy legördülő menü, amelyben a kimeneti formátumot adhatod meg. A számos lehetőség közül most csak a LaTeX- és az XML-lehetőségeket emelném ki. A legalsó vezérlőelem első fülén láthatod a program kimenetét. A második fül egy periódusos rendszert foglal magában, a harmadikon háromdimenziós függvényeket rajzolhatsz, míg a negyediken paraméteres egyenletek jeleníthetők meg grafikonon.

A nyelv alapjai

A rendszer programozható. Saját nyelve van, amely nagyon könnyen elsajátítható. Sőt, ha valaki ismeri egy kicsit a C-t, annak a HartMath ebből a szempontból semmi újat nem fog mutatni.

Mindenekelőtt meg kell értened a numerikus és a szimbolikus kiértékelés közti különbséget. Ha egy kifejezést numerikusan értékelünk ki, csak a számeredményt kapjuk meg, még akkor is, ha az esetleg nem pontos (akár a számológép). Ellenben a szimbolikus kiértékelésnél a végtelen szakaszos tizedes törtek megmaradnak két szám hányadosaként, az irracionális számok nem kerülnek „kiszámolásra" stb. A legjobb példa a gyök kettő, amelyről bizonyított, hogy irracionális. Numerikus kiértékelés esetén az `Sqrt(2)` parancsra az 1,4142135... eredményt kapjuk, szimbolikus esetén viszont gyök kettőt.

Az alapműveletek a C-hez hasonlóan a +, -, *, / jelekkel végezhetők el. A hatványozás jele a ^, a faktoriálisé a !. Zárójelek is használhatók, például a (2^3)/(2-4+5*3) kifejezést szimbolikusan kiértékelve a 8/13 eredményt kapjuk. Lehetőség nyílik változók használatára. Így az x=2; y=3; x*y kifejezés értéke 6. Összevont műveleti jelekkel is dolgozhatunk: x/=2 (ami egyenértékű az x = x / 2 kifejezéssel). Használhatunk függvényeket is, többek között a gyökvonást, a szinuszt és koszinuszt, a vektorok értelmezése pedig csak ezek segítségével oldható meg. Egy változó törlése is egyedül függvénnyel lehetséges: `Clear(x)`.

Jelentősebb függvények

- `Print(arg)`
  A megadott argumentumot kiírja a képernyőre, és Null értékkel tér vissza.
- `Print( Szia vilÆg! )`
- `Div(arg1,arg2)`
  Az `arg1` és `arg2` maradékos osztásából származó egész rész.
- `Div(13,8) -> 1`
- `Mod(arg1,arg2)`
  Az `arg1` és `arg2` maradékos osztásából származó maradék.
- `Mod(13,8) -> 5`
- `Log(arg)`
  Az `arg` természetes alapú logaritmusa.
- `Log(E^3) -> 3`
- `Sqrt(arg)`
  Az `arg` négyzetgyöke, például `Sqrt(25) -> 5`.
- `PieChart(lista)`
  „Tortadiagramot” rajzol egy újabb fülre a listában megadott értékek alapján. A lista elemei { és } kapcsos zárójelek között helyezkednek el, és vesszővel vannak elválasztva, például `PieChart({1,2,3})`.
- `IsProbablePrime(arg)`
  A JVM `BigInteger.isProbablePrime` megvalósítás alapján nagy valószínűséggel meg tudja állapítani, hogy prím-e, például `IsProbablePrime(63) -> False`.
- `Fibonacci(n)`
  Meghatározza a Fibonacci-sorozat n-edik elemét, például a `Fibonacci(3333) ->` kifejezésre egy egész képernyőt betöltő számot kaptam, méghozzá meglepően gyorsan.
- `SolveP(polynom,valt)`
  A legfeljebb másodfokú polinomot megoldja a változóra, például a `SolveP(x^2,x)` kifejezés az x^2=0 egyenlet gyökeit keresi. Eredményül a {0,0} listát kapjuk, ebből látszik, hogy egyetlen megoldása van.

Programozni jó

A HartMath saját nyelve is tartalmazza az összes eset+szétválasztásos módszert és ciklusszervező eljárást, én mégis csak kettőt mutatnék be. A nyelv primitívekből építkezik, így egy elágazás nem több, mint egy függvény a megfelelő értékekkel ellátva.

```
If (allitas, igaz_ag, hamis_ag,
➥egyeb_ag)
```

Ha az `allitas` igaz, az `igaz_ag` értékelődik ki, ha hamis, akkor a `hamis_ag`, és ha logikailag nem értelmezhető az `allitas`, akkor pedig az `egyeb_ag`.

```
If(22,Print( Igaz ),Print( Hamis ),
➥Print( nem logikai kifejezØs! ))
➥  nem logikai kifejezØs!
If(True,Print( Igaz ),Print( Hamis ),
➥Print( nem logikai kifefejezØs! ))
➥-> Igaz
For (elott,feltetel,utan,kifejezes)
```

Az `elott` a ciklusba történő belépés előtt fut le. A `feltetel` minden iteráció elején, és ha hamis lesz, a ciklus nem fut tovább. Az `utan` minden iteráció végén értékelődik ki. A `kifejezes` a ciklusmag.

```
For (i=0,i<10,i++,Print(i))
➥-> 0 1 2 3 4 5 6 7 8 9
```

Ezek alapján készítsünk egy olyan programot, ami 2-től 100-ig írja ki a prímszámokat. Nyilván szükség lesz egy számlálós ciklusra. Ezen belül ellenőrizzük, hogy a ciklusváltozó prím-e, és ha igen, írjuk ki a képernyőre.

```
For (i=2,i<100,i++,If(IsProbablePrime
➥(i),Print(i),Null,Null))
```

A `Null` értékekre azért van szükség az `If()`-ben, mert a mezőket nem lehet üresen hagyni, még akkor sem, ha az adott ágba semmit sem akarsz semmit.

Függvények, fájlok használata

A HartMathban saját függvények írására is lehetőség nyílik. Egy jól megírt függvény nem baj, ha megmarad a merevlemezen, és később vissza lehet tölteni. Ezt a lehetőséget egy példán keresztül szeretném bemutatni.

```
Clear(x); kobre(x_)=x^3
```

Ez a kifejezés létrehoz egy `kobre()` függvényt, amely az átadott értéket köbre emeli. Ki is próbálhatod:

```
kobre(2) -> 8
```

Fontos, hogy a függvény változói a függvény meghatározása előtt más kifejezésekben nem használhatók. Ezért szerepel egy `Clear()` utasítás még az elején. Láthatod azt is, hogy a függvény változói az értéklistában egy aláhúzásjellel bővülnek.

Van már egy jól működő függvényünk, de ha most kilépsz a programból, elveszik. Ha későbbre is meg szeretnéd őrizni, a mentésre használd a `WriteObject()` függvényt. Ezzel nemcsak függvényeket, hanem bármilyen kifejezést, például egy előző számítás végeredményét tartalmazó változót is menteni lehet. Az így mentett állományok a *<saját könyvtár>/.hartmath/hmscripts* könyvtárba kerülnek.

```
WriteObject( kobos ,kobre(x_)=x^3)
```

Ezzel a fenti könyvtárban létrehozol egy *kobos.hm* különleges formátumú fájlt. Ezt a `ReadObject()`-tel lehet visszatölteni:

```
ReadObject( kobos )
```

Végezetül

Mindent egybevéve a HartMath nagyon jó rendszer. Látszik rajta, hogy nem érte még el az 1.0-s változatot, mert a kézikönyvben vannak olyan függvények, amelyek a programban nem működnek, viszont üzembiztos, így a mindennapi használatra is nyugodtan ajánlhatom.

Fülöp Balázs (xut@freemail.hu) 17 éves, imádja a Túró Rudit, a Debian Linuxot és a teheneket. Az ELTE Radnóti Miklós Gyakorlóiskola tanulója immár ötödik éve. Kedvenc írója Slawomir Mro□ek. Leginkább a számítógépes hálózatok biztonsága érdekli.