**Calcul Simbolic pentru rezolvarea unor tipuri de ecuații cu parametri**

Pelle Remus-Nicolae – remusnicolaepelle@gmail.com

Lect. univ. dr. Laslo Eugen – lasloeugen@yahoo.com

Facultatea de Științe, Universitatea din Oradea

**ABSTRACT:** *În această lucrare, ne propunem construirea unei aplicații vizuale care să vină în ajutorul atât a profesorilor meditatori, cât și a elevilor, în ceea ce privește rezolvarea asistată a unei plaje cât mai mari de probleme specifice examenelor de Bacalaureat. Aplicația permite introducerea unor ecuații cu parametri și construiește rezolvarea în funcție de aceștia pentru diferite categorii de probleme, cum ar fi: polinoame, determinanți, derivate, ecuații ș.a.m.d.*

**1. Introducere**

Pentru realizarea acestui proiect, am creat o clasă de bază, numită Symbol, în care orice număr real de forma (v = valoare, pv = putere valoare, s = simbol, ps = putere simbol) este scris ca și o matrice .

Pentru a putea folosi aplicația, trebuie să înțelegi cum funcționează programul și să reușești să adaptezi scrierea unei ecuații pentru a fi recunoscută de program.

În aplicație, pentru astfel de numere, sunt acoperite atât operațiile de bază (cum ar fi adunarea și înmulțirea), cât și operații mai avansate (radicali, determinanți, derivate, etc).

**2. Dezvoltarea aplicației la nivel structural**

Aplicația are la bază 3 clase publice, Symbol prezentată mai sus, Number și Equation. Despre ultimele două, puteți afla informații la capitolul 3. Pentru fiecare dintre acestea, mai există câte o clasă statică în care sunt implementate operațiile ce se pot realiza între obiecte ale clasei.

Pentru a simplifica unele unele lucruri, am creat o enumerație de simboluri:

|  |
| --- |
| public enum Symbols { Null, x,y,z}; |

Clasa Symbol cuprinde 6 constructori pentru a putea fi instanțiată mai ușor în unele cazuri. Constructorul principal primește, ca și parametri, 2 valori int, un Symbols și încă un int, fiind asociate în ordine pentru a da valori lui v, pv, s și ps. Există constructori pentru cazurile în care dai doar valoarea lui v sau doar valorile lui v și pv, sau cazul în care dai doar valoarea lui s sau valoarea lui s împreună cu valoarea lui ps, și cazul în care dai ca și parametru un Symbol, valorile acestuia fiind copiate în noul Symbol creat.

Pentru a putea folosi aplicația, trebuie să înțelegi codul și să poți transforma o ecuație pentru a fi înțeleasă de program. De exemplu, pentru 54x2y3z2, trebuie să creăm un Number care conține aceste elemente:

|  |
| --- |
| Number number = new Number();  number.values.Add(new Symbol(5, 4, Symbols.x, 2));  number.values.Add(new Symbol(Symbols.y, 3));  number.values.Add(new Symbol(Symbols.z, 2)); |

Dacă dorim să încercăm să adăugăm 3xy5z3, trebuie să creăm un Equation care să conțină primul Number, și un al doilea Number nou creat:

|  |
| --- |
| Number n2 = new Number();  n2.values.Add(new Symbol(3, 1, Symbols.x, 1));  n2.values.Add(new Symbol(Symbols.y, 5));  n2.values.Add(new Symbol(Symbols.z, 3));  Equation e = new Equation();  e.values.Add(n1);  e.values.Add(n2); |

După care se folosesc operațiile de înmulțire pentru Number si de adunare pentru Equation, și se vor afișa elementele cât mai simplificat posibil.

Pentru afișare, funcția ToString() trebuie rescrisă. Pentru Symbol, aceasta implică un string care, la început, este format doar din “(“. Se adaugă valoarea v, după care se verifică dacă pv este diferit de 1, caz în care adăugăm “^”,pv și “)“, iar în caz contrar, se mai adaugă doar “)“. După care se verifică dacă s este diferit de “Null”, adăugăm “(“ și s, iar daca ps este diferit de 1, adăugăm “^”, ps și “)“, în caz contrar, se mai adaugă doar “)“. De exemplu, pentru 76x3, se va afișa (7^6)(x^3).

La Number și Equation, funcția ToString() este mult mai ușor de scris. La Number, avem un string care concatenează toate elementele din acel Number. La Equation, avem un string care concatenează toate elementele, iar între acestea, vom pune “+”.

**3. Concepte folosite**

Pentru realizarea operațiilor între două numere cu simbol, simple metode sunt suficiente. Dar pentru simplificarea scrierii unor ecuații complexe, este nevoie de o succesiune de numere păstrate în memorie de către calculator. Pentru a realiza acest lucru, este necesară o listă de astfel de numere.

Am decis că va fi mai ușor să avem elementele ce trebuiesc înmulțite între ele listate separat de elementele ce trebuiesc adunate. Un număr dintr-o ecuație poate fi de forma 2xyz, care s-ar reprezenta ca și o listă de trei Symbols ce urmează să fie înmulțite:

Cum putem avea mai multe elemente de acest tip ce vor urma să fie adunate, am decis să realizez clasele astfel: o clasă Number ce conține o listă de Symbol, elemente ce vor urma să fie înmulțite între ele, și o altă clasă Equation ce conține o listă de Number, ce urmează să fie adunate.

|  |
| --- |
| public class Number  {  public List<Symbol> values = new List<Symbol>();  // ...  }  public class Equation  {  public List<Number> values = new List<Number>();  // ...  } |

**CLASA “Number”**

Clasa Number are ca scop realizarea operațiilor de înmulțire dintr-o ecuație ce trebuie simplificată. Elementele ce urmează să fie înmulțite sunt salvate ca o listă de Symbol.

Pentru a reduce cât mai mult dimensiunea unui Number, sunt tratate niște cazuri excepționale, care duc la simplificarea scrierii.

* Idempotența lui 1: 1pv sau v0 sunt scrise ca și 11, lucru care rezultă în mai puține înmulțiri.
* Neutralitatea lui 0: dacă întalnim 0pv, tot Number este redus la un singur element, 0.
* Transformarea în termen liber: vpvs0 va fi scris ca și vpv, lucru care poate duce la eliminarea unui simbol din ecuație.

După ce s-au realizat simplificările, se face înmulțirea efectivă a elementelor rămase. Valorile reale împreună cu primul simbol apărut vor fi păstrate în primul element din listă, iar pentru fiecare simbol nou, vom aloca un nou element în listă. Pentru numerele reale, se face înmultirea, iar la simboluri, se adună exponenții.

**CLASA “Equation”**

Clasa Equation are ca scop realizarea operațiilor de adunare dintr-o ecuație ce trebuie simplificată. Elementele ce urmează să fie adunate sunt salvate ca o listă de Number.

Pentru ca această clasă să decidă dacă se poate face adunarea între elemente, este necesar un mod în care să deosebească elementele asemenea. Din moment ce un element poate fi format din mai multe simboluri, o metodă mai eficientă decât să se verifice succesiv fiecare element la fiecare adunare, este să salvăm în ordine alfabetică simbolurile împreună cu puterea acestora într-un string. De exemplu, pentru 3x2yz, stringul după care se decide dacă se poate aduna cu alte elemente, este “x2y1z1”. După determinarea tuturor stringurilor, elementele vor fi comparate. Cele care au același string, sunt elemente asemenea între care se poate face adunare.

**4. Rezultate și concluzii**

În această lucrare, ne-am propus dezvoltarea unei aplicații care să rezolve o multitudine de probleme ce conțin ecuații cu parametri, care necesită rezolvări pe diferite categorii, cum ar fi: polinoame, determinanți, derivate, etc. Aceste probleme se folosesc predominant în pregătirea pentru examenul de Bacalaureat.

Rezultatele obținute sunt promițătoare, se pot realiza operații de bază între două numere, se poate simplifica scrierea unei colectii de numere ce urmează sa fie înmulțite și/sau adunate, operațiile mai complexe sunt, la rândul lor, funcționale, iar pe viitor, se vor putea rezolva ecuații complexe într-o multidudine de feluri, fiind un suport semnificativ pentru elevi, studenți, și chiar și pentru profesori.