Universitatea Tehnica din Cluj-Napoca

Facultatea de Automatica si Calculatoare

Sectia Calculatoare si Tehnologia Informatiei

Documentatie pentru aplicatie de procesare a polinoamelor

Student: Timotei Molcut

Cluj-Napoca

21 Martie 2019

Cuprins

1. Obiectivul temei
2. Analiza problemei, modelare, scenarii, cazuri de utilizare
3. Proiectare
4. Implementare
5. Rezultate
6. Concluzii
7. Bibliografie

Obiectivul temei

Principalul obiectiv al acestei teme este de a procesa polinoamele. Ce inseamna, de fapt, acest lucru? Prin aceasta tema se doreste posibilitatea ca un utilizator al acestei aplicatii sa poata introduce un string, care reprezinta un polinom, iar pe acest polinom sa se poata efectua operatii aritmetice, precum, adunare, scadere, inmultire si impartire. Pe langa acestea se pot efectua operatiile de derivare si integrare asupra polinoamelor.

Pentru a putea atinge acest obiectiv, este nevoie de trasarea anumitor obiective secundare. Printre acestea se numara procesarea stringului introdus de utilizator(parse), crearea unei interfete grafice care sa usureze lucrul utilizatorului si, desigur,organizarea orientata pe obiect a conceptului de polinom.

Procesarea stringului de intrare(sir de caractere), consta in a ajuta sistemul de procesare sa inteleaga ce se afla in acel string. Noi, oamenii, suntem fiinte inteligente care sunt capabile sa inteleaga direct ce anume se codifica in stringul respectiv. Spre deosebire de noi, computer-ul nu este in stare de acest lucru. Pentru el, acel string nu reprezinta altceva decat o succesiune de biti de ‚0’ si ‚1’. Asadar, procesarea stringului, denumita in engleza „parse”, este o necesitate pentru aplicatia de fata si va fi detaliata in capitolele urmatoare.

Interfata grafica este, de asemenea, un factor care contribuie la o functionare, orientata spre utilizator, adica utilizatorul va putea sa interactioneze cu aplicatia mult mai usor. Aceasta interfata grafica este reprezentata de o fereastra pop-up care contine mai multe butoane si campuri de introducere a polinomului sub forma de string. Prin apasarea butoanelor se pot obtine rezultate la operatiile dorite. In capitolele urmatoare se va intra in detaliu in prezentarea acestei interfete grafice.

In final, cel mai important aspect secundar este felul in care se va organiza obiectual conceptul de polinom. In cazul de fata, s-a folosit abordarea conceptului mai restrans de monom, iar apoi s-a dezvoltat in cazul conceptului mai complex de polinom. Metoda de stocare si felul in care se va prelucra polinomul in anumite operatii este cheia acestui obiectiv. Si pentru acest aspect se vor aduce mai multe explicatii in capitolele urmatoare.

Analiza problemei, modelare, scenarii, cazuri de utilizare

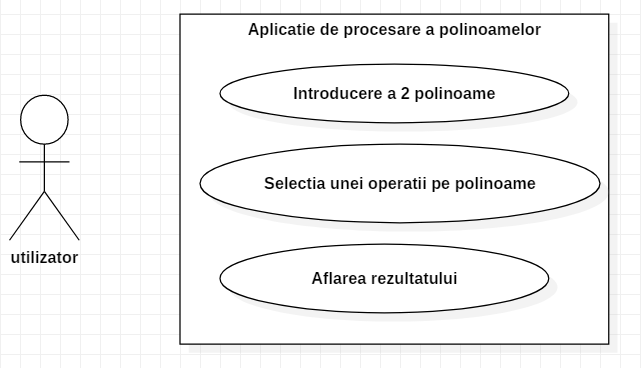
Analiza acestei probleme determina modalitatea de organizare in programare orientata pe obiect a conceptului de polinom.

Pentru inceput este nevoie de o impartire a problemei in subprobleme mai mici. Daca incepem analiza din punct de vedere teoretic ne putem pune intrebarea: „Ce este un polinom?”. In matematica, un polinom reprezinta o expresie construita in general dintr-o variabila si mai multe constante, care se leaga prin operatii de adunare, scadere, inmultire si ridicare la o putere pozitiva constanta, de regula intreaga. Asadar, un polinom poate arata in felul urmator: poli.PNG. De fapt, se poate spune ca un polinom se poate impartii in unitati mai mici numite monoame. Asadar, am reusit sa impartim problema noastra in cazuri mai mici. Acest monom este caracterizat de o singura inmultire intre o constanta si o variabila ridicata la o putere pozitiva si constanta, iar polinomul va reprezenta o succesiune de monomame adunate cu semnele lor(+ sau -). In consecinta, putem sa modelam problema dupa conceptul de monom si polinom.

Scenariul de functionare este determinat de utilizarea a doua polinoame, introduse de catre utilizator, care vor fi folosite la efectuarea operatiilor principale, de tip binar, adica: adunare, scadere, inmultire, impartire. In cazul derivarii si integrarii, operatia este unara fiind nevoie doar de un polinom. Astfel se poate selecta unul dintre polinoamele pe care dorim sa efectuam operatia de derivare/integrare.

Utilizarea acestei aplicatii necesita introducerea polinomului intr-un anumit standard. Adica procesarea stringului introdus va functiona doar daca polinomul este introdus asa cum trebuie. Intuitiv, am ales ca monomul(la fel si polinomul) sa fie introdus cu coeficient, ales in cazul de fata de tip intreg, lipit de caracterul ‚x’. Dupa caracterul ‚x’ urmeaza caracterul ‚^’ care reprezinta ridicarea la putere, iar imediat dupa acest caracter urmeaza exponentul, ales tot de tip intreg. In cazul in care stringul nu are acest model de reprezentare a monomului, nu se va putea efectua parsarea si se va anunta o eroare de parsare.

Cerintele functionale ale acestei aplicatii sunt: introducerea a doua polinoame alese de utilizator, alegerea unei operatii de efectuat pe polinoamele respective si afisarea rezultatului. Aceste cerinte se pot reprezenta intr-o diagrama use-case precum cea de mai jos.



Prima cerinta este posibilitatea de a introducere doua polinoame in interfata grafica. Aceste doua polinoame sunt parsate si convertite in tipul Polinom. A doua cerinta(selectarea operatiei) se realizeaza prin apasarea unuia dintre butoanele interfetei. Fiecare buton are o operatie corespunzatoare care se va aplica asupra polinoamelor existente la cerinta precedenta. Dupa aflarea rezultatului, acesta se va afisa in interfata. Practic, utilizatorul trebuie sa dea niste date si va primi in schimb alte date.

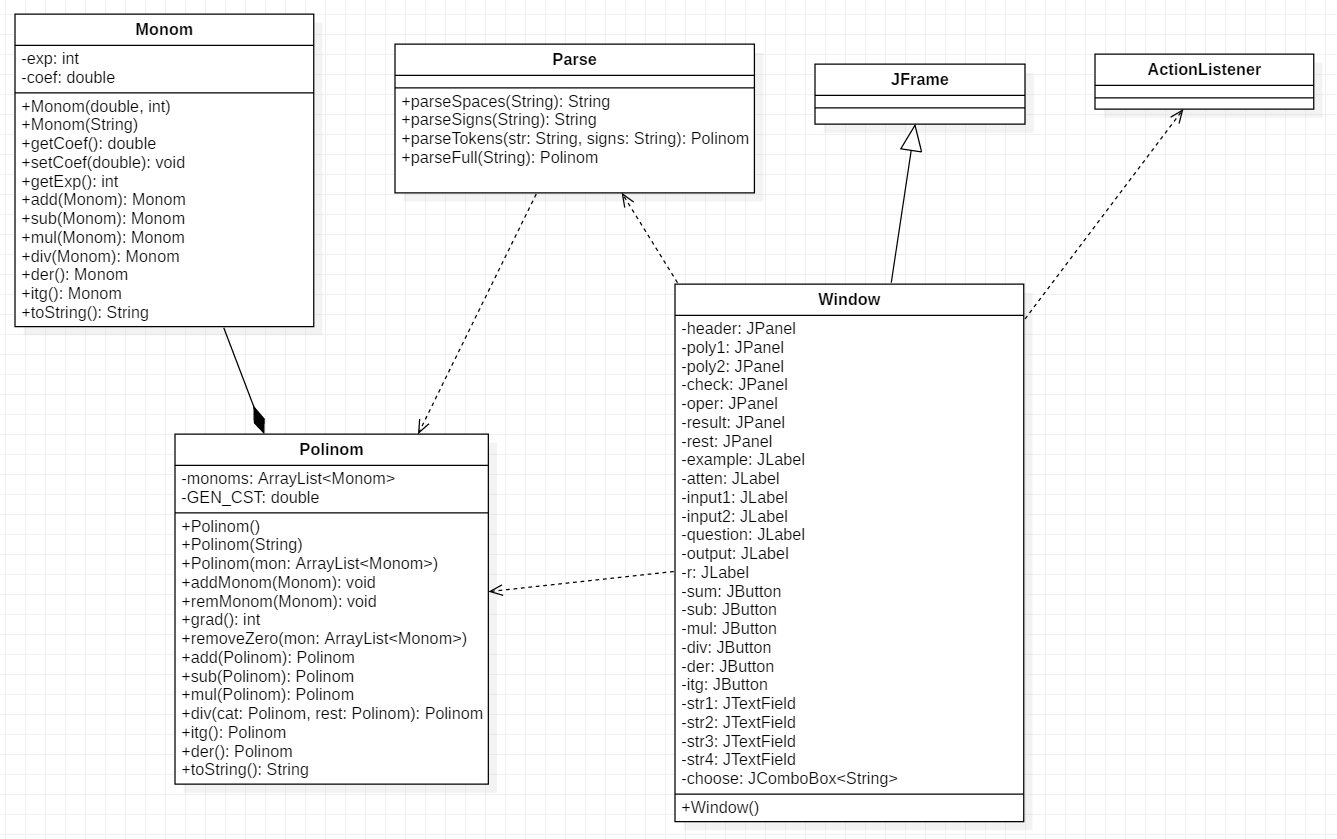
Proiectare

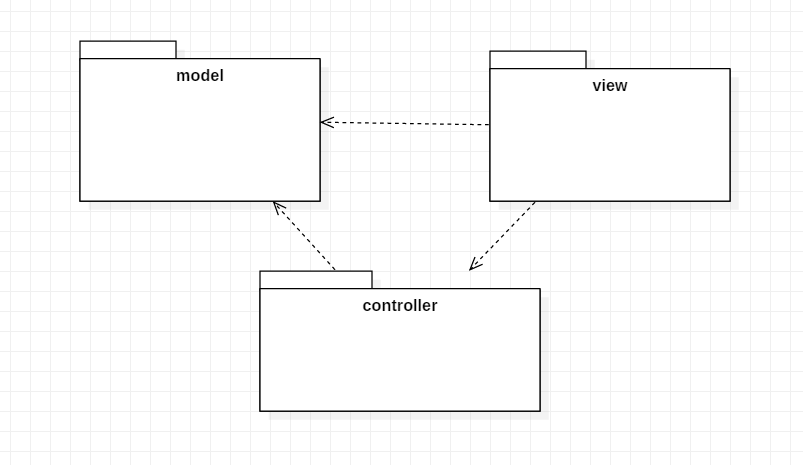
Dupa cum am precizat deja, conceptul de Polinom l-am redus la conceptul de Monom. Acest lucru este foarte important, deoarece implementarea operatiilor pe Polinom va fi determinata de implementarea operatiilor pe Monom. De asemenea, Polinomul este de fapt o lista de Monoame. Prin urmare structura de date folosita este ArrayList<Monom>, adica o lista de tipul Monom. Asadar, Polinomul va contine unul sau mai multe Monoame, determinandu-se o relatie de compozitie. Despre Monom se poate spune ca este suficienta codificarea cu un atribut pentru coeficient si inca unul pentru exponent.

Pentru a obtine un Polinom dintr-un String este nevoie de o procesare pe care am ales sa o fac prin metodele din clasa Parse. Parsarea va da ca rezultat un Polinom asadar, se determina o relatie de dependenta intre clasa Parse si clasa Polinom(implicit si Monom).

In final, interfara grafica are nevoie si de parsare si de calcularea operatiilor pe Polinoame. Asadar, va exista inca o relatie de dependenta intre clasa Window, care implementeaza interfata, si clasele Parse si Polinom.

Toate aceste relatii se pot evidentia prin diagramele UML de clase si de pachete de mai jos.





Dupa cum se observa, in pachetul model am ales sa pun clasele Monom si Polinom, in pachetul controller, clasa Parse, iar in pachetul view, clasa Window.

In clasa Window am ales setarea de ActionListener pentru butoanele existente pentru a usura codul si modul de implementare, deoarece in metoda suprascrisa pentru fiecare buton, actionPerformed(), folosesc atribute private ale clasei Window. Astfel, codul poate fi inteles si mai bine.

In general nu am folosit algoritmi deja existenti, ci pentru fiecare operatie am incercat, si cred ca am reusit in anumita masura, sa scriu un cod potrivit pentru operatiile necesare.

Implementare

In randurile urmatoare voi descrie felul in care am implementat clasele deja amintite si metodele care implementeaza operatiile necesare.

Incepand cu nivelul cel mai de jos al complexitatii, voi descrie clasa Monom. Dupa cum am mai spus, pentru a reprezenta un monom ca structura de date este nevoie doar de doua atribute si anume: coeficientul si exponentul sau. Se cere ca acest coeficient sa fie intreg, dar deoarece la integrare ar putea sa existe o impartire este mai potrivit sa se foloseasca tipul double, pentru o reprezentare cat mai precisa. Desigur ca rezolvarea absoluta ar fi ca acest coeficient sa aiba un tip generic care sa poata avea valoarea de tip „int” sau „double”. De cealalta parte, exponentul are tip intreg tot timpul. In cod am descris doi constructori: unul pentru doua valori de tip „double” si „int”, si celalalt pentru un String care este convertit, pe bucati(adica este impartit cu metoda split()), prin metoda Integer.parseInt(...). La partea de operatii am scris sase metode fiecare pentru una dintre operatiile necesare. De exemplu, voi descrie metodele de inmultire si integrare.

Metoda de inmultire returneaza un nou Monom care va avea coeficientul egal cu coeficientul primului monom(this) inmultit cu coeficientul celui de-al doilea monom(trimis ca parametru). Pentru noul exponent se va trimite suma exponentilor operanzilor.

Metoda de integrare va returna un Monom potrivit doar in cazul in care exponentul este natural, deoarece pentru un exponent negativ integrarea ar da un logaritm natural care nu ar avea sens drept polinom. Asadar noul coeficient va fi egal cu raportul dintre coeficientului curent si exponentul adunat cu ‚1’, iar exponentul nou va fi cel vechi adunat cu ‚1’. Constanta de integrare va fi introdusa in cadrul polinomului.

Trecand la un nivel mai complex, voi descrie implementare clasei Polinom. Dupa cum am mai spus, Polinomul este o lista de Monoame. Asadar, este nevoie doar de un singur atribut: un ArrayList<Monom>. Deoarece integrarea necesita o constanta, am declarat in plus o variabila de tip „static final double”, numita GEN\_CST. Drept constructori am ales mai multe tipuri: un constructor fara parametru, unul pentru o lista de monoame deja populata si inca unul pentru un String.

Prima metoda imprtanta implementeaza adunarea. Deoarece doua polinoame care se aduna pot sa nu aiba acelasi grad, este nevoie de verificarea acestui caz si introducerea in lista polinomului final a monoamelor cu grad mai mare, de la unul dintre polinoame. Dupa aceasta, fiecare dintre monoamele polinomului se aduna pe rand folosind metoda implemntata pe monoame. La final daca raman zero-uri la puterile mai mari, acestia se scot din lista, deoarece reprezinta redundanta.

Metoda de scadere se implementeaza aproximativ la fel ca cea de adunare cu deosebirea ca aceasta nu este comutativa. Deci va conta ordinea de scadere.

Metoda de inmultire este putin mai simpla. Fiecare monom din primul monom se va inmulti cu un monom din al doilea polinom. In lista rezultata vor exista monoame cu acelasi exponent care trebuiesc grupati intr-o alta lista, care va fi finala.

Metoda de impartire este de departe cea mai complexa operatie. Aceasta necesita inca un polinom pentru rest. Asadar, restul l-am trimis ca parametru pentru metoda, ca sa fie modificat in interiorul ei, iar catul a fost returnat de catre metoda. De asemenea, am avut grija de cazul in care un polinom cu grad mai mic se imparte la un polinom cu grad mai mare(ceea ce ar da un raport imposibil de reprezentat ca polinom). Asadar, impartirea se efectueaza doar daca primul polinom are gradul mai mare. Restul trebuie sa fie mai mic decat impartitorul. Prin urmare am folosit o bucla „while” in care restului ii scad din starea curenta polinomul rezultat din inmultirea impartitorului cu un polinom unar dat de impartirea monoamelor de grad maxim. Acest polinom unar va determina treptat forma catului.

Metoda pentru integrare parcurge lista curenta de monoame si le integreaza pe fiecare(folosind metoda de la monoame), iar la final se adauga un monom de grad ‚0’, adica constanta de integrare, care va avea valoarea GEN\_CST inmultita cu un numar „random”.

Metoda de derivare parcurge fiecare monom din lista curenta si il deriveaza, iar rezultatul se pune intr-o lista noua(la fel ca la restul metodelor).

Finalizand cu partea de model, ajung la controller, adica la clasa Parse. Pe aceasta clasa am implementat-o cu patru metode interconectate: parseSpaces(), parseSigns(), parseTokens() si parseFull().

Metoda parseSpaces() doreste sa elimine spatiile posibil existente in String-ul de intrare. De asemenea aici se semnaleaza daca cumva utilizatorul a introdus in exponent negativ(x^-2).

Metoda parseSigns() parcurge stringul de intrare pentru a afla semnele fiecarui monom. De ce este nevoie de aceasta metoda? Deoarece token-izarea ce urmeaza se va face pe langa semnele monoamelor, adica token-urile nu vor mai contine semne. Astfel, este utila memorarea tuturor semnelor.

Metoda care are o importanta majora este parseTokens(). Dupa cum am zis, aceasta metoda imparte String-ul de intrare in token-uri despartite de semn. Dupa aceasta se foloseste constructorul Monom(String) pentru fiecare token.

Metoda finala, care le inglobeza pe celelalte este parseFull(). Aceasta primeste String-ul dorit si folosindu-se de metodele de mai sus va returna un Polinom.

Pentru partea de view, am creat clasa Window(subclasa a lui JFrame). Acest window contine mai multe JPanel: header, poly1, poly2, check, oper, result, rest. Fiecare dintre acestea contine JLabel, JTextField, JButton. Fereastra Window are o organizare de GridLayout(7, 1), ceea ce inseamna ca voi putea pune sapte JPanel-uri, adica exact cate folosesc. Aproape fiecare JPanel foloseste o organizare FlowLayout, adica componentele se vor pune in ordine. Doar panoul pentru operatii, oper, va avea GridLayout(2, 3), pentru a pune cele sase butoane ale sale pe doua linii.

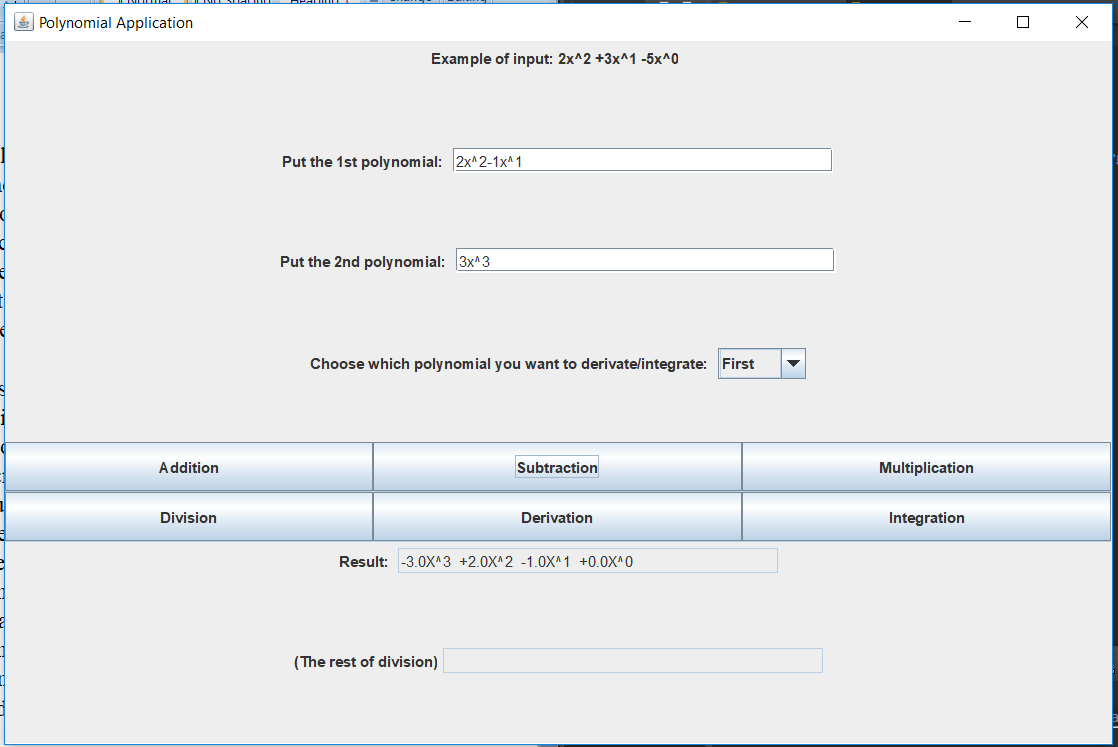
Intr-unul dintre aceste panouri este un JComboBox care selecteaza pe care dintre polinoame se va face operatia derivare/integrare.

In panourile poly1 si poly2 se vor introduce polinoamele, iar in panourile result si rest se va afisa rezultatul opeartiei si restul impartiri, daca este cazul.

Partea care da viata interfetei reprezinta butoanele, pentru care se adauga cate un ActionListener(). Pentru fiecare ActionListener(), se suprascrie metoda actionPerformed(), potrivita pentru cazul respectiv. De exemplu, pentru apasarea butonul de adunare, se vor parsa stringurile introduse in doua polinoame, care se vor aduna prin metoda add(), iar rezultatul se va afisa din nou in interfata. Acest procedeu este aproximativ le fel si pentru restul butoanelor.

Rezultate

La rularea aplicatie aceasta va arata in felul urmator.

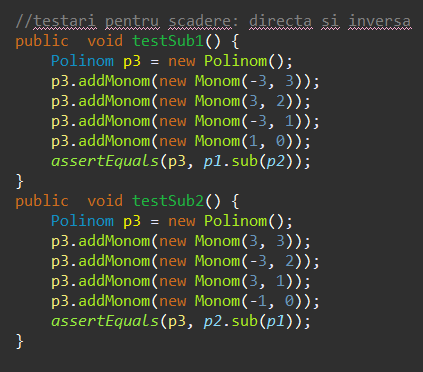


Dupa cum se observa, s-au introdus deja doua polinoame si s-a selectat operatia de scadere(Subtraction). Asadar, „2x^2-1x^1 – 3x^3” va da „-3x^3 +2x^2 -1x^1 +0x^0”.

Rezultate testare cu Junit

In clasa TestPolinom am scris cate doua metode de testare pentru fiecare operatie. Asadar, in total sunt 14 metode de testare. In fiecare caz de testare am folosit metoda assertEquals(), ceea ce a necesitat suprascrierea metodei equals() in clasele Monom si Polinom. Modul de testare este urmatorul:creez doua polinoame(direct cu constructor), iar al treilea polinom va fi rezultatul operatiei corespunzatoare(pe acest polinom il creez eu). In final, apelez metoda assertEquals() pentru polinomul rezultat si polinomul creat de metoda operatiei pe primele doua polinoame.

Testarea pentru scadere arata in felul urmator. Testarea directa inseamna, primul polinom, operatie, al doilea polinom, iar cea inversa este vice-versa.



Toate testarile pe care le-am scris functioneaza corect.

Concluzii

In concluzie, as putea spune ca aceasta tema m-a indemnat sa rezolv anumite lucruri pe care nu le-am mai facut pana acum. Spre exemplu: parsarea String-urilor, la care recunosc ca as avea nevoie de mai multa pregatire. De asemena, gasirea algoritmului necesar pentru operatiile pe polinoame a fost o alta provocare la gandire si, nu in ultimul rand, realizarea interfetei grafice a fost o cerinta legata de modul de organizare vizuala prietenoasa cu utilizatorul.

Dezvoltarile ulterioare la care m-am gandit ar fi: tratarea tuturor cazurilor de String introdus de catre utilizator, folosirea de tip generic pentru clasa Monom, posibila optimizare a codului operatiilor. De asemenea, o alta cerinta ar fi ca polinomul sa se poata evalua intr-un numar dat de catre utilizator. O cerinta de nivel mai ridicat, ar fi ca polinomul introdus sa se deseneze grafic.

Bibliografie

* World Wide Web
* <https://en.wikipedia.org/wiki/Main_Page>
* <https://www.tutorialspoint.com>
* <https://stackoverflow.com>