Evaluarea de expresii Matematice

Îndrumător:
dr. ing. Daniel Morariu

Student:
Barna Alin Vasile Anul 2 C
Grupa 222/2

Cuprins

1 Specificarea cerinţelor software

1.1 Introducere

Principalul rol al aplicației este evaluarea de expresii matematice, cu implementarea operatoriilor „+”, „-”, „*”, „/”, „( )”, ... și functii de genul „sin”, „cos”, etc.

Expresiile se vor introduce in forma normala sau din fişier. Avand posibilitatea de a vedea ultimele expresii evaluate.

1.1.1 Obiective

- Evaluarea expresiilor matematice cu operatorii „+”, „-”, „*”, „/”, „( )”, „^”.
- Implementarea funtiilor „sqrt (x)”, „ln (x)”, „exp (x)”, „lg (x)”,
- Implementarea functiilor trigonometrice directe „sin”, „cos”, „tg”, „ctg” şi rezolvarea lor atat prin paramentru de tip radian cat şi prim paramentru cu tip grad.
- Implementarea functiilor trigonometrice inverse „asin”, „acos”, „atg”, „actg” şi afisarea rezultatelor sale atat in radian cat şi in grade.
- Formarea a doua moduri de lucru, unul clasic/standard, in care utilizatorul poate evalua o singura expresie. Şi un al doile mod („more epressions=mai multe expresii”) in care utilizatorul poate evalua mai multe expresii in acelaş timp. Al doile mod permite utilizatorul a citi mai multe expresii din fişier.
- Posibilitatea de a vedea ultimele expresii evaluate, de a putea prelua in modul standard una din ultimele expresii evaluate, sau de a prelua toate expresiile din istoric in modul „more expressions”.
- Tratarea tuturor erorilor printre care:
    - **Impartirea la zero**. Tratarea acestei erorii printr-un mesaj catre utilizator care sa il avertizeze asupra impartirii cu zero.
    - **Spatiile** lasate in mod eronat in interiorul expresiei.
    - **Semne şi simboluri eronate.** Tratarea erorii printr-un mesaj catre utilizator.
    - **Lipsa parantezei deschise ‚(’.** Mesaj de eroare.
    - **Lipsa parantezei inchise ‚)’.** Mesaj de eroare.
    - **Scrierea unei expresii incorecte.** Mesaj de eroare.

1.1.2 Scop

Scopul acestei aplicatii este de a realiza o aplicatie care sa permita evaluarea de e expresii matematice, cu implementarea operatoriilor „+”, „-”, „*”, „/”, „( )”, „^”,... şi functii de genul „sin”, „cos”, etc.. Expresiile se vor putea introduce in forma normala sau din fişier. Aplicatia va avea posibilitatea de a vedea ultimele expresii evaluate prin intermediul History..

1.1.3 Definiţii, Acronime şi Abrevieri

Toate clasele proprii sunt definite cu majuscule: EXPRESSION, CFILE, RWS.

Denumirea elementelor din cadrul aplicatiei au o denumire semnificativa, urmand o regula care va fi prezentata mai jos.

O prescurtare importanta in cadrul aplicariei este RWS = Read Write Solve. Am folosit aceasta prescurtarea atat pentru clasa respectiva cat și pentru variabilele de tipul acestei clase.

REGULA DE DEFINIRE A ELEMENTELOR

Toate elementele din interiorul unei forme au fost definite sub urmatoarea regula:

nume_formaDeCareApartine_utilitate

nume: nume implicit sau prescurtarea numelui sau initiala semnificativa

Exemple: Button -> Btn_, Edit -> Edit_, Label -> Label_ / L_ , etc.

formaDeCareApartine: numele/ prescurtatrea/ initiala(initialele) semnificative formei din care fac parte.

Exemplu: Butoanele din forma Calculator „FormCalculator" -> BtnC_

utilitate: ceea ce face elementul sau pentru ce este el folosit.

Exemple: Butoanele care introduc cifre -> BtnC1, BtnC2, etc, BtnCNrPI etc.

Edit-ul in care este afisata expresia/rezultatul -> EditCExp / EditCRes

Caracterul ‚_'

1.1.4 Tehnologiile utilizate

Borland C++ Builder 6

Cerinţe specifice

Citirea / Scrierea in fişier. Expresiile pot fi citite şi din fişier, iar dacă se doreste se pot scrie dupa evaluarea lor in alt fişier in care va aparea atat expresia cat şi rezultatul dat. Avem şi posibilitatea schimbarii fişierelor de citire sau scriere.

Forma Poloneza Postfixata. Notaţie matematică care ajuta foarte mult la posibilitatea evaluarii unei expresii matematice de catre un calculator.

Evaluarea expresiei. Evaluarea unei expresii care se realizeaza cu ajutorul formei poloneza.

Istoricul. Posibilitatea de a vedea ultimele expresii evaluate şi de a relua dacă se doreste una sau mai multe din aceste expresii pentru posibila reevaluarea a lor.

Modul „More Expressions”. Avem posibilitatea de a citi pana la 10 expresii din fişier si de a le evalua deodata. Deasemenea expresiile se pot introduce şi de utilizator.

Tratarea erorilor. Pentru anumite cazuri primim mesaj de eroare şi cateva indicatii care pot fi cauzele erorii.

2 Prima Funcţionalitatea

2.1 Forma Poloneza

Aducerea expresii care urmează să fie evaluată în forma poloneză necesară în această aplicaţie, altfel evaluarea expresiei ar fi imposibilă.

Forma poloneză postfixată este o notaţie matematică în care fiecare operator (funcţie) urmează dupa operandul său. Forma poloneză a unei expresii ajută la evaluarea în mod corect din punct de vedere a ordinii de evaluare a operatoriilor şi a funcţiilor.

Exemple:

2 + 3 -> in forma poloneză postfixată -> 2 3 +
sin(30) -> in forma poloneză postfixată -> 30 sin
2 + 3 - 1 -> in forma poloneză postfixată -> 2 3 + 1 -

2.2 Fluxul de evenimente

2.2.1 Fluxul de bază

Această funcţionalitate este defapt o funcţie *„formaPoloneza ( ) ”* din clasa EXPRESSION, care se foloseşte de membrul *„expression”* al aceleaşi clase şi de alte funcţii din interiorul clasei pentru atingerea scopului.

Utilizatorul are un rol important în pornirea funcţionalităţii, prin apăsarea unui buton de evaluare ‚=’, el trimite expresia care urmează a fi adusă în forma poloneză. Rezultatul acestei funcţionalitaţi nu este desponibil utilizatorului, aplicaţia nu afişează forma poloneză şi nu permite accesul la ea .

Prototipul funcţiei: ***std::queue<string> formaPoloneza()***.Se foloseşte în mod necesar de membrul „*expression”* al clasei, astfel ea nu primeste niciun parametru. La introarcerea din funcţie se returnează o coada *„queue”* de string-uri, care vazută ca o coadă reprezintă forma poloneză. Rezultatul putea fi pus şi într-un string, dar pentru uşurinţa evaluării mai departe a expresiei am ales această variantă (coada).

In interiorul functiei am mai initializa o variabila locala de tip stiva de string-uri in care voi adauga temporar operatorii si functiile.

Modul de realizare a formei poloneze pas cu pas.

1. **Parcurgem şirul caracter cu caracter.**
2. **Dacă este număr adăugăm adică adăugam in coada.**

   Verificare dacă este numar am facut-o astfel:

   -am verificat dacă caracterul curent este cifra, apoi atat timp cat urmatorul caracter este tot cifra sau este punct , . ’ le adaug intr-un string care mai tarziu va fi convertit in double.

3. **Dacă este functie adaug numele functiei in stiva de operatori.**

Prima data verific dacă am *log(b,x)* deoarece are o implementare mai diferita si am preferat tratarea ei separat.

```
if(eLog(i)) {
stivaOperator.push("log"); i+=2;
 i++;//deoarece urmatorul caracter ar trebui sa fie '(' si sarim peste
//extragem primul numar (baza) si il adaugam in forma poloneza string str;
while(eCifra(expression[i]) || expression[i]=='.'){
 str=str+STR(expression[i]);
i++;
}
 i--; poloneza.push(str);
//acum ar urma caracterul ',' care delimiteaza numarul
//il extragem si il adaugam in stiva
str=""; //resetam stringul while(eCifra(expression[i]) || expression[i]=='.'){
str=str+STR(expression[i]); i++;
}
 i--; poloneza.push(str);
i++;//deoarece urmatorul caracter era ')' am sarit peste }
```

Exemplu: log(2 , 0.5) → 2 0.5 log

Pentru restul functiilor am implementat tot in cadrul clasei, metode la care trimit paramentru pozitia curenta din *expression*, adica *,i',* metodele sunt private si imi returneaza bool, astfel stiu dacă am sau nu functie.

4. **Altfel dacă este paranteză deschisă , (' adaugam caracterul , (' in stiva de operatori**

5. **Dacă este paranteza inchisa ,)', extragem din *stiva* de operatori și adăugam la coada *ploneza* pana dam de paranteza deschisa , (' . Scoatem paranteza ,(' din stiva.**

6. **Dacă este operator: atat timp cat operatorul din varful *stivei* are prioritate mai mare decat operatorul curent, adaugam in forma *poloneza* elementul din varful *stivei*.**

Pentru verificarea prioritatii operatorilor am implementat o metoda care imi returneaza un int si care imi spune care operator este mai mare.

7. **Apoi se introduce aperatorul curent in *stiva*.**
8. **Se adauga toti operatorii ramasi pe *stiva* in forma *poloneza*.**

Exemplu: 6+(9^4-2)*sin(0)

(1)Parcurgem sirul caracter cu caracter: primul caracter ‚3'

(2)	Poloneza: 6	; Stiva:	- *urmeaza ‚+'*
(6)	Poloneza: 6	; Stiva: +	- *urmeaza ‚('*
(4)	Poloneza: 6	; Stiva: + (	- *urmeaza ‚9'*
(2)	Poloneza: 6 9	; Stiva: + (	- *urmeaza ‚^'*
(6)	Poloneza: 6 9	; Stiva: + (^	- *urmeaza ‚4'*
(2)	Poloneza: 6 9 4	; Stiva: + (^	- *urmeaza ‚-'*
(6) +(7)	Poloneza: 6 9 4 ^	; Stiva: + (-	- *urmeaza ‚2'*
(2)	Poloneza: 6 9 4 ^ 2	; Stiva: + (-	- *urmeaza ‚)'*
(5)	Poloneza: 6 9 4 ^2 -	; Stiva: +	- *urmeaza ‚*'*
(8)	Poloneza: 6 9 4 ^2 -	; Stiva: + *	- *urmeaza „sin"*
(3)	Poloneza: 6 9 4 ^ 2 -	; Stiva + * sin	- *urmeaza ‚('*
(4)	Poloneza: 6 9 4 ^ 2 -	; Stiva: + * sin (	- *urmeaza ‚0'*
(2)	Poloneza: 6 9 4 ^2 - 0	; Stiva: + * sin (	- *urmeaza ‚)'*
(5)	Poloneza: 6 9 4 ^2 - 0	; Stiva: + * sin	
(8)	Poloneza: **6 9 4 ^ 2 – 0 sin * +**	; Stiva: goala	

6+(9^4-2)*sin(0) → 6 9 4 ^ 2 – 0 sin * +

2.2.2 Pre-condiţii

Utilizatorul trebuie să introduca o expresie matematica corecta. Nu trebuie sa introduca simboluri care nu pot fi interpretate (ex: #, @, etc...), nu trebuie sa ultilizeze moduri neclare pentru aplicatie de scriere a expresiei (ex: ln5 in loc de ln(5)), trebuie să aiba o atenţie deosebită la folosirea parantezelor.

2.2.3 Post-condiţii

Utilizatorul nu are acces la rezultatul acestei funcţionalităţi. Această functionalitate este un ajutor la evaluarea expresiei.

3 A doua Functionalitatea

3.1 Evaluare expresiei

Posibilitatea evaluarii unei expresii este scopul principal al aplicatiei.

3.2 Descriere

Evaluarea expresiei presupune calcularea unei expresii in mod corect: trebuie sa tina cont de prioritatea fiecarui operator, sa identifice si sa rezolve functiile.

3.3 Fluxul de evenimente

3.3.1 Fluxul de baza

Posibilitatea evaluarii unei expresii se face in cadrul functiei ***double EXPRESSION::evaluare().*** Un rol important in evaluarea expresiei il joaca forma poloneza. Pentru inceput am declarat o variabila locala de tip queue <string> in care se pune forma poloneza a expresiei care doreste sa fie evaluata.

queue <string> coada=formaPoloneza();

Voi prezenta pas cu pas modul de decurgere a functionalitatii.

Pentru realizarea algoritmului de evaluare am nevoie de o stiva de tip double, care sa imi pastreze termenii si rezultatele partiale care vor devein termeni in urmatoarele operatii.

stack <double> stiva;

Pe scurt in cateva cuvinte pot principiul algoritmului de evaluare. Exemplul dat anterior la forma poloneza era: **3+(9^4-2)*sin(0)** care a ajuns in forma poloneza astfel: **3 9 4 ^ 2 – 0 sin * +.**

Deci coada arata astfel : **3 9 4 ^ 2 – 0 sin * +**

- → Cat timp am de scos numere, le scot din coada si le pun in stiva.
- → Daca am un operator binar, scot din stiva ultimi doi termeni, fac operatia respectiva intre ei iar rezultatul il pun iar pe stiva.
- → Daca am functie sau operator unar, scot de pe stiva ultimul termen, aplic functia sau operatorul una asupra lui, iar rezultatul la fel il pun pe stiva.
- → Se repeta acesti pasi pana cand coada este goala.
- → La sfarsit ce se afla in varful stivei va fi rezultatul pe care trebuie sa il returnam.

In continuare voi prezenta si codul pentru a putea detalia modul de operationare a functionalitatii.

Voi elimina partiile din cod care le consider inutile pentru explicarea si intelegerea functionarii. Majoritatea partiilor de cod excluse vor fi parti de cod care se repeta de mai multe ori pentru fiecare functie sau operator.

3.3.2 Pre-conditii

Utilizatorul are rolul de a introduce o expresie pe ecran sau sa citeasca o expresie din fisier. Tot utilizatorului ii revine rolul de a cere solicitarea rezultatului prin apasarea unui buton de evaluare „=”. Dar un alt lucru important este ca utilizatorul sa scrie o expresie corecta din punct de vedere matematic si din punct de vederea al aplicatiei.

Daca utilizatorul doreste folosirea functiilor trigonometrice se atrage atenti asupra parametrului trimis de utilizator, daca este in radian sau grade. Exista posibilitatea comutarii din radian in grade si inver, dar utilizatorul este cel care are acces sa i-a decizia folosirii functiilor trigonometrice in radian sau grade.

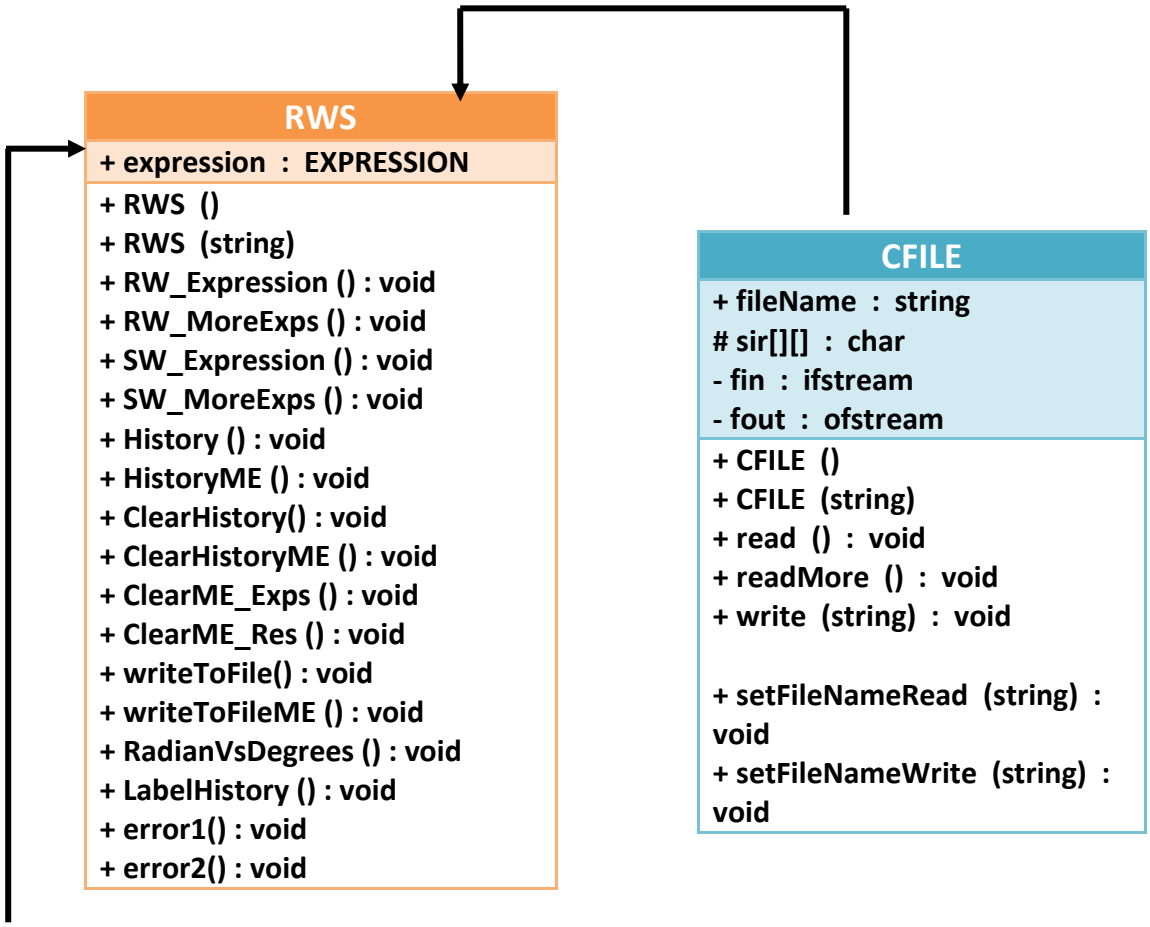
3.3.3 Post-conditii

Utilizatorul primeste pe ecran rezultatul evaluarii expresiei sau a expresiilor, dar se poate astepta si la mesaje de eroare atunci cand expresia introdusa nu este corecta, iar acel caz a fost tratat.

Din cauza netratarii tuturor cazurilor de eroare, utilizatorul se poate astepta la blocarea aplicatiti.

4 Implementare

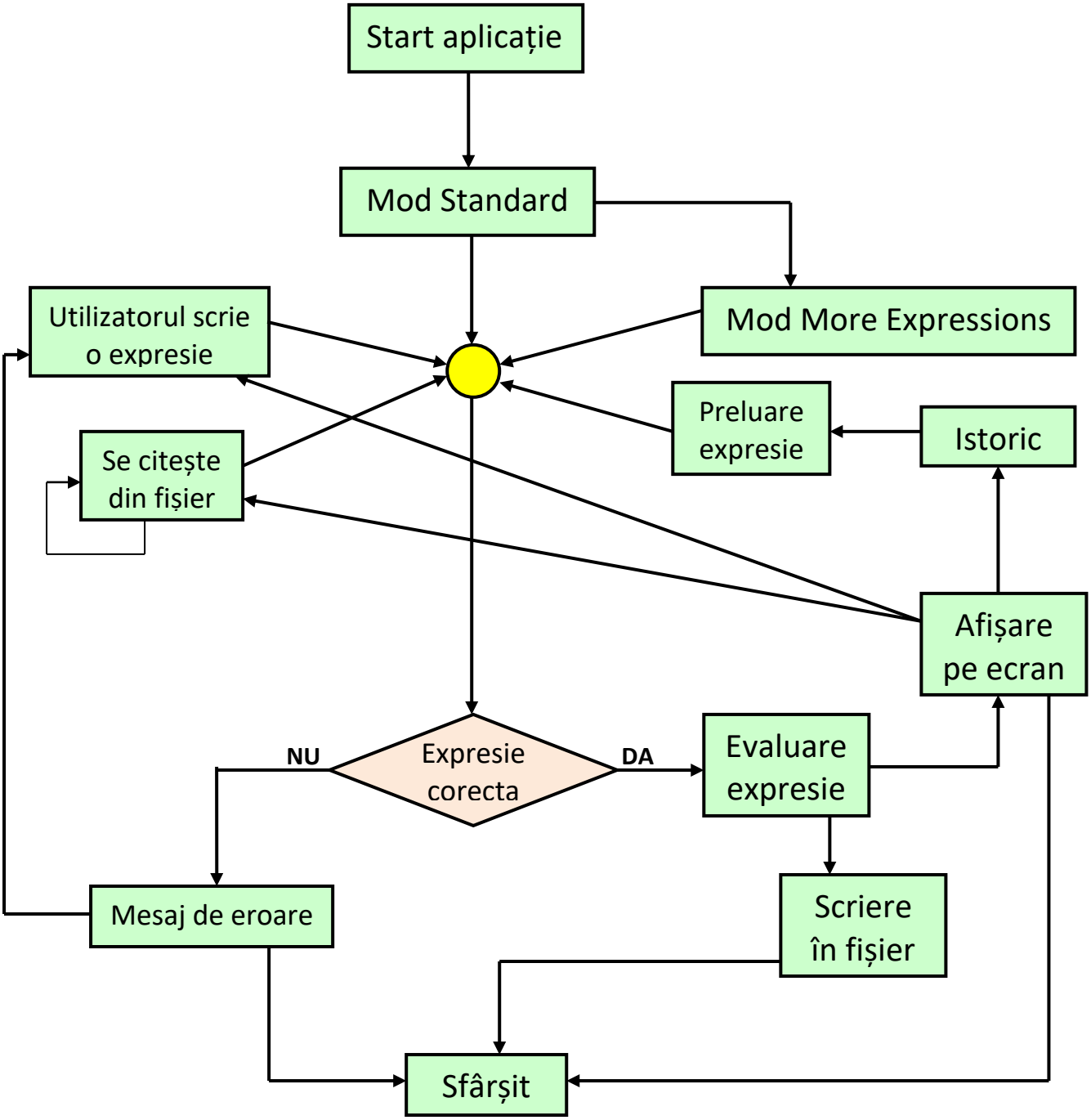
4.1 Diagrama de clase



EXPRESSION		
+expression : string +radian : bool		
+EXPRESSION ()	-eLn (int) : bool	-eAsin (int) : bool
+EXPRESSION (string)	-eLn (string) : bool	-eAsin (string) : bool
+evaluare () : double	-eLg (int) : bool	-eAcos (int) : bool
-formaPoloneza () : queue<string>	-eLg (string) : bool	-eAcos (string) : bool
	-eLog (int) : bool	-eAtg (int) : bool
- prioritateOp (string) : int	-eLog (string) : bool	-eAtg (string) : bool
-eOperator (char) : bool	-eSqrt (int) : bool	-eActg (int) : bool
-eCifra (char) : bool	-eSqrt (string) : bool	-eActg (string) : bool
-cifra (char) : double	-eSin (int) : bool	-eNrPI (int) : bool
-numar (string str) : double	-eSin (string) : bool	-eNrPI (string) : bool
-eFunctie (int) : bool	-eCos (int) : bool	-PDeschisa (char) : bool
-eFunctie (string) : bool	-eCos (string) : bool	-PInchisa (char) : bool
-ePow (int) : bool	-eTg (int) : bool	-compara (string,char) : bool
-ePow (string) : bool	-eTg (string) : bool	-STR (char) : string
-eExp (int) : bool	-eCtg (int) : bool	-CHAR (string) : char
-eExp (string) : bool	-eCtg (string) : bool	

4.2 Descriere detaliată

Schema logică:



5 Bibliografie

- Functiile trigonometrice
  http://www.cplusplus.com/reference/cmath/

- Pentru citire din fisier
  http://www.cplusplus.com/reference/istream/

- Pentru scrierea in fisier
  http://www.cplusplus.com/reference/ostream/

- „Programare Orientata pe Obiecte. Principii c++”, Macarie Breazu 2002