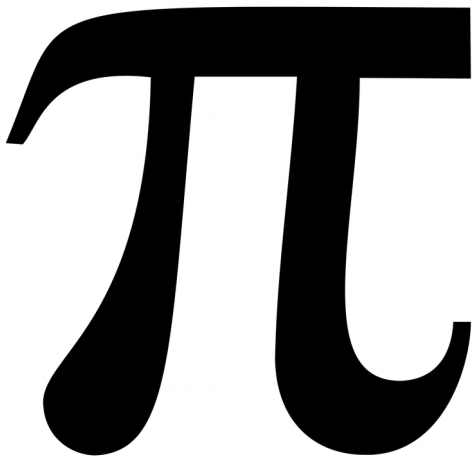
PROIECT PENTRU OBȚINEREA ATESTĂRII PROFESIONALE ÎN INFORMATICĂ



Titlul lucrării:

Elemente de algebră și analiză matematică

Colegiul Național „Vasile Alecsandri”, Galați

Profesor coordonator:
Luminița Mihaela Cobzaru

Elev: **Tiftilov Vladislav**
Clasa a-XII-a B

Mai 2020

Cuprins

Motivația alegerii temei

De ce matematica?

„Matematica este o limbă și o știință.”
Lucian Blaga

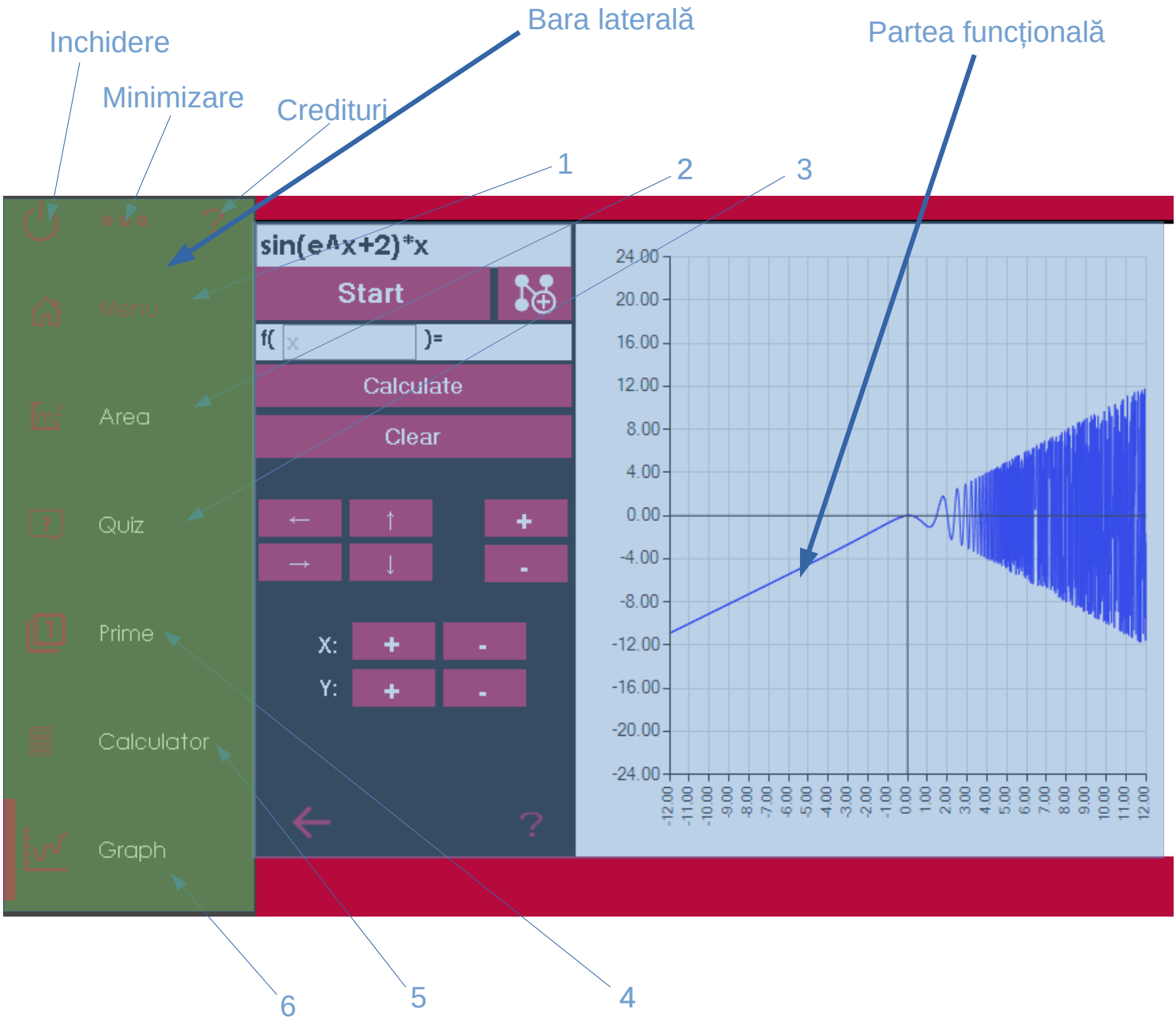
Necesitatea de a realiza relații sociale a determinat ca omul să-și alcătuiască propriile instrumente. Primul și cel mai important instrument a fost și rămâne comunicarea, non-verbală sau prin intermediul unei limbi.
În domeniul informaticii, în prezent există peste 250 limbaje de programare. Fiecare din acestea au la baza sa un limbaj universal, care a apărut cu mult înaintea conceperii termenului de programare, aceasta este matematica.
Matematica reprezintă un limbaj universal, utilizat în absoluta majoritate a tuturor domeniilor existente. Universalitatea matematicii este determinată în primul rând prin postulatele care stau la baza matematicii. De-asemenea, strictețea matematicii nu permite încălcarea regulilor bine definite, de-aceea matematica este aceiași în orice limbă și în orice țară.
Argumentul principal pentru alegerea matematicii ca temă de lucru reprezintă universalitatea sa dar și multitudinea de informație accesibilă.
De asemenea, am ales acest domeniu din cauza provocărilor pe care pe care le poate provoca matematica. Astfel în cât abordarea chiar și a celei mai simple probleme matematice se poate transforma în ore de algoritmizare și aplicarea într-un limbaj de programare.

Scopul lucrării

Scopul lucrării este cel de a crea o platforma accesibilă pentru rezolvarea și vizualizarea problemelor matematice.
Un alt motiv este cel de a dezvolta competențe în abordarea algoritmică a unei probleme matematice și implementării într-un limbaj de programare.
De asemenea, aplicația pe care am dezvoltat-o reprezintă un produs cu o aplicabilitate practică atât pentru elevi cât și studenți. Astfel încât, aplicația dată a fost dezvoltată din considerentul unei posibile modificări dar și a adăugării noilor caracteristici.

Interfața aplicației

Interfața grafică a aplicației reprezintă un obiect de tip formă, împărțit în 2 părți, bara laterală și partea funcțională în care sunt alternate obiecte de tip **User Control**.

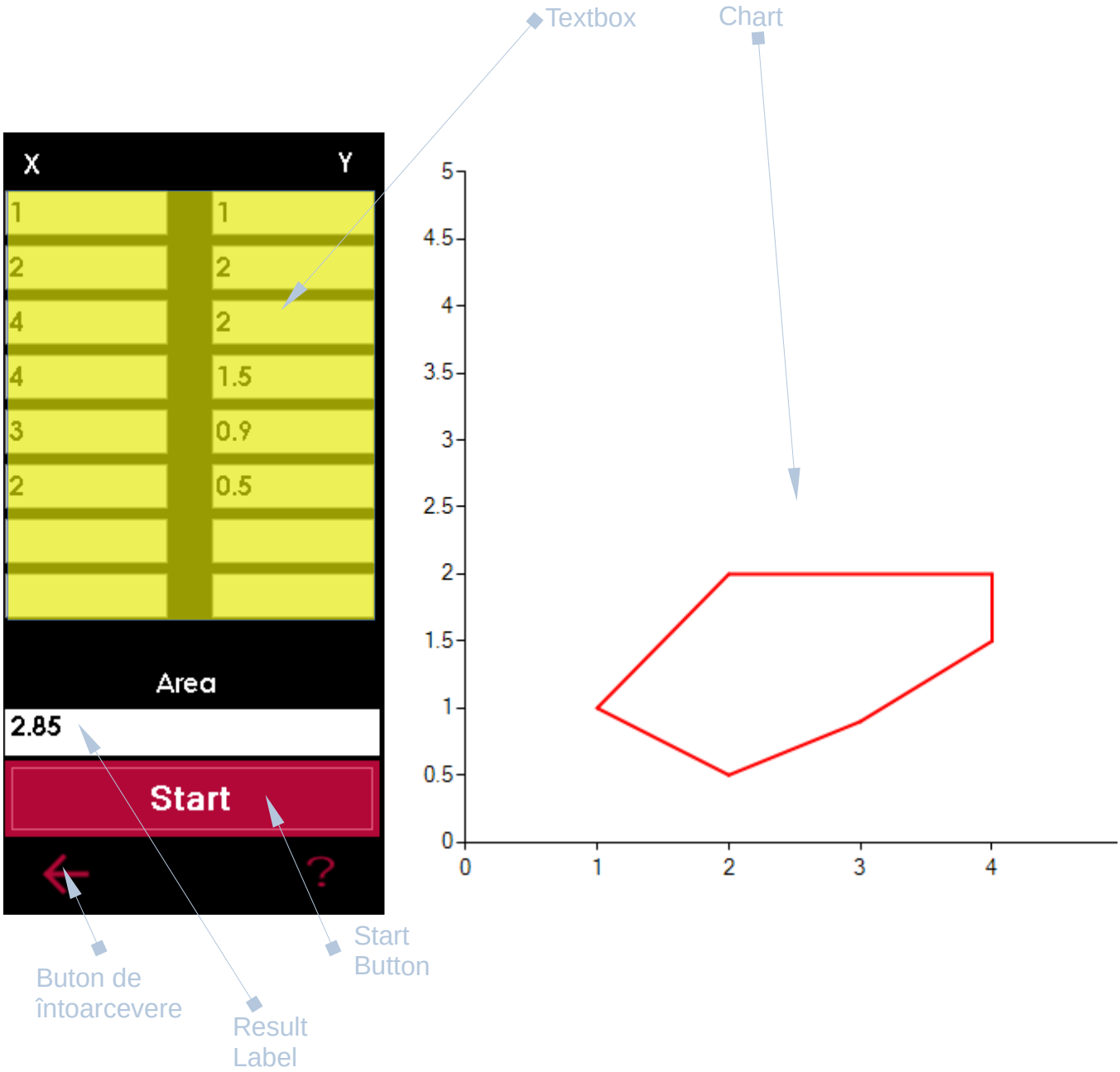


1. Menu

Pagina de start a aplicației fără funcționalitate.

2. Area

Este constituit dintr-un obiect de tip **Chart**, **textbox**-uri în care se introduc coordonatele carteziene a unui poligon simplu, un **label** rezultat în care este afișat aria figurii, **butonul Start** și **butonul de întoarcere** la geamul precedent.



La apăsarea **butonului de întoarcere** obiectul curent (**user control**) devine invizibil și este trimis în fundal.

Prin apăsarea butonului **Start** se apelează funcția **get_points()** care transformă textul din **textbox**-uri în valori de tip **double** care sunt salvate în doi vectori **x**,**y** pentru abscisele și ordonatele corespunzătoare figurii geometrice.

```
private void get_points()
        {
            try
            {
                if (textBox1.Text != "" && textBox9.Text != "")
                {
                    x[n] = Convert.ToDouble(textBox1.Text);
                    y[n] = Convert.ToDouble(textBox9.Text);
                    n++;
                }
             ...
            }
            catch{}
         }
```

În continuare, punctele salvate în cei doi vectori sunt adăugate în obiectul **chart1** cu denumirea seriei de „**Gf**“, care este de tip linie.
La pasul următor este apelată funcția **polygonArea(x,y,n)**, care returnează aria poligonului format cu ajutorul formulei lui **Shoelace**.

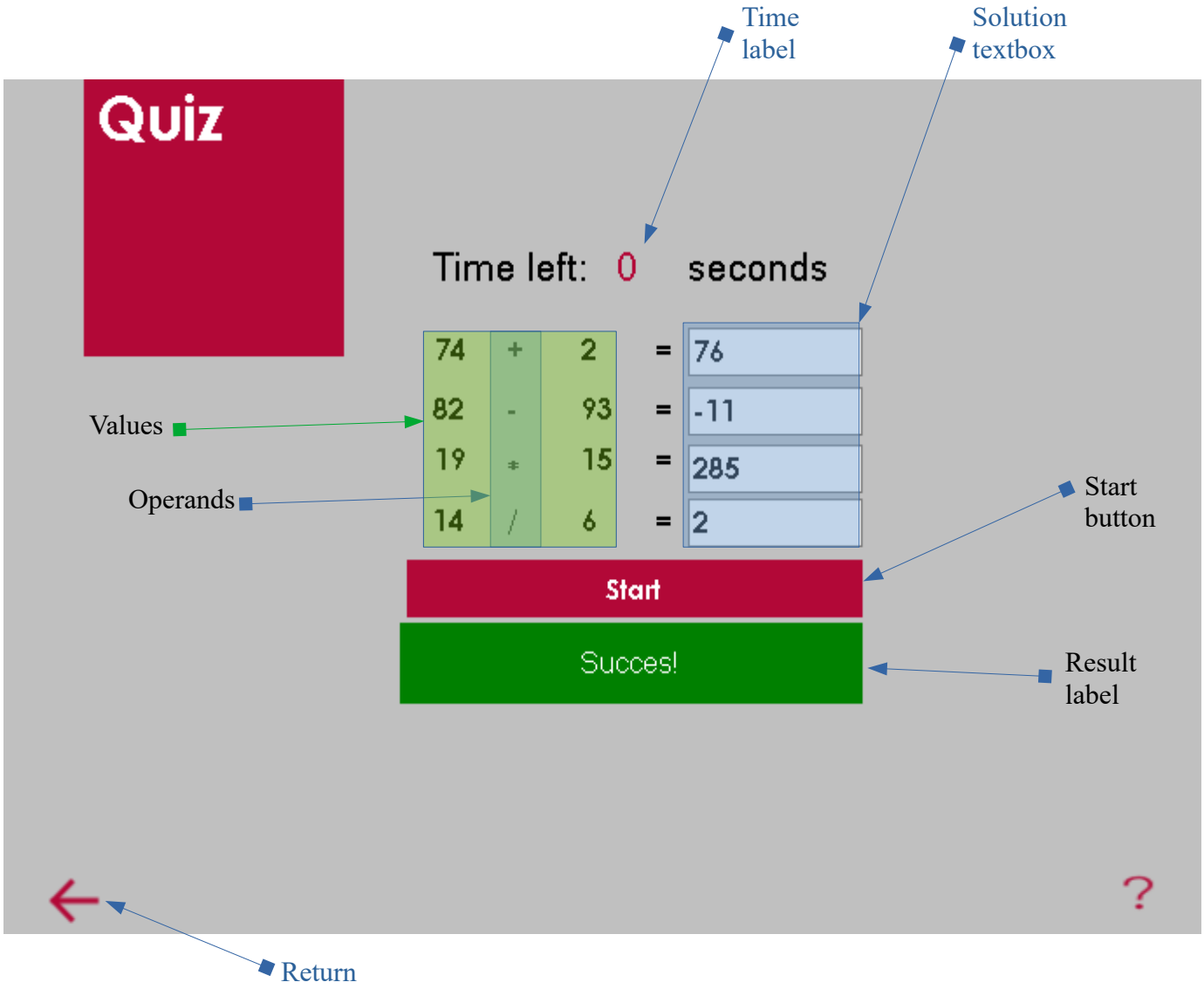
```

private double polygonArea(double[] X, double[] Y, int n)
        {
            double area = 0.0;
            int j = n - 1;
            for (int i = 0; i < n; i++)
            {
                area += (X[j] + X[i]) * (Y[j] - Y[i]);
                j = i;
            }
            return Math.Abs(area / 2.0);
        }

```

3. Quiz

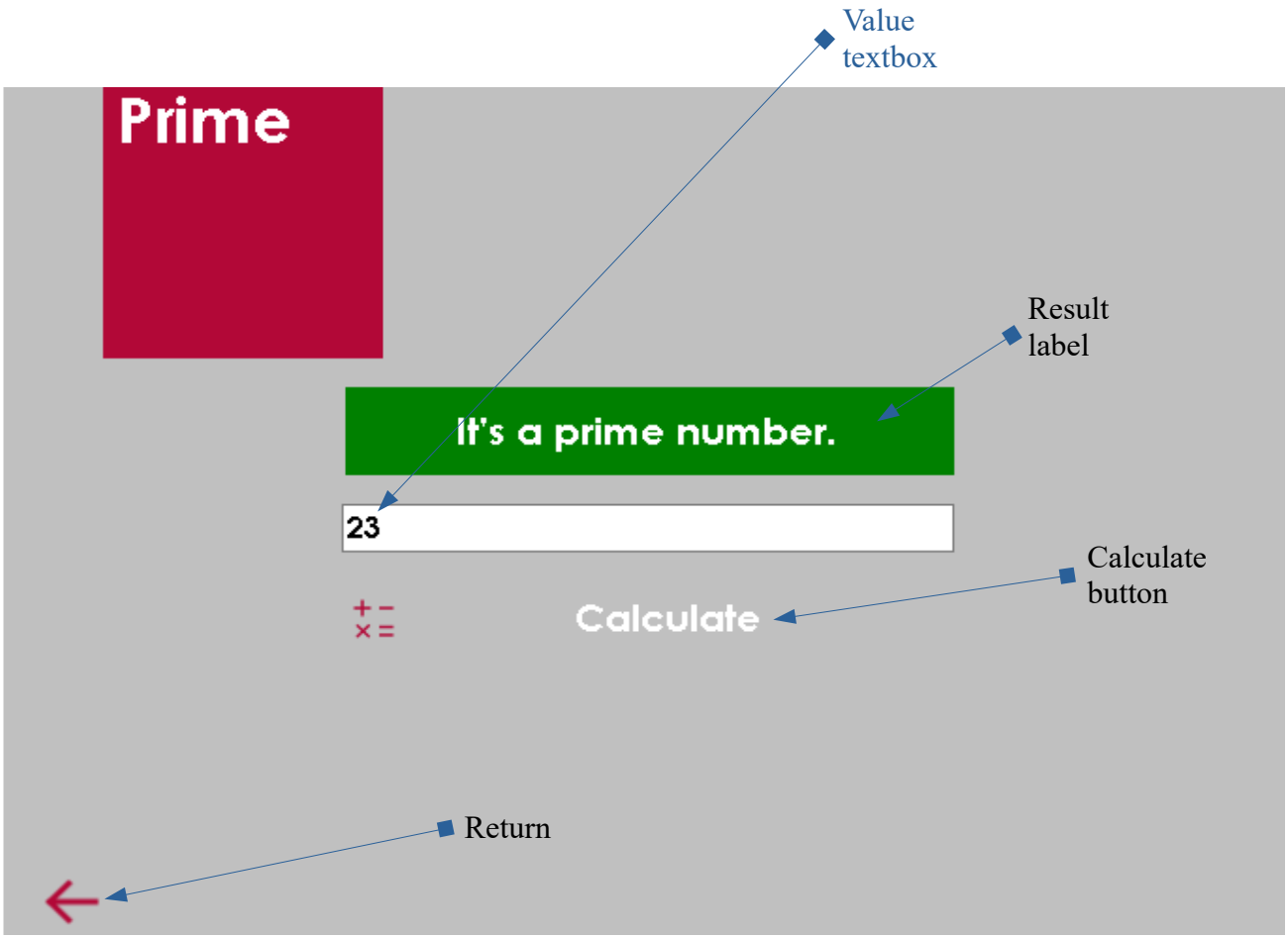
Este un **user control** compus din **label**-uri **Values** în care sunt generate valori de la 0 la 100, **label**-uri **Operands** care determină operația dintre cele două **label**-uri **Values** adiacente, **textbox**-uri **Solution** în care sunt scrise rezultatul presupus a expresiei din stânga, butonul **Return** (similiar cu cel anterior),label **Time** care afișează timpul rămas, label **Result** în care se afișează rezultatul corespunzător calculelor și butonul **Start**.



La apăsarea butonului **Start** obiectul **timer1** este activat, la fiecare tick al său (**1 tick/s**) valoarea numerică a label-lui **Time** este redus cu **1**. În momentul când valoarea dată este egală cu **0,** sau dacă se apasă butonul **Finish** sunt verificate valorile introduse în textbox-uri cu ajutorul funcției **check()**, care returnează valoarea **1** dacă toate soluțiile sunt corecte sau **0** pentru cazul opus, sau **2** în cazul în care are loc o excepție (nu este introdusă soluția, soluția are un format necorespunzător). În dependentă de valoarea returnată a funcției **check()**, label-lui **Result** ia proprietățile corespunzătoare rezultatului returnat.

```
1 public int check()
2         {
3             int ok = 1;
4             try
5             {
6                 if (int.Parse(textBox1.Text) != int.Parse(lab1.Text) + int.Parse(lab2.Text))
7                     ok = 0;
8                 if (int.Parse(textBox2.Text) != int.Parse(lab3.Text) - int.Parse(lab4.Text))
9                     ok = 0;
10                if (int.Parse(textBox3.Text) != int.Parse(lab5.Text) * int.Parse(lab6.Text))
11                    ok = 0;
12                if (int.Parse(textBox4.Text) != int.Parse(lab7.Text) / int.Parse(lab8.Text))
13                    ok = 0;
14            }
15            catch
16            { ok = 2; }
17            return ok;
18        }
```

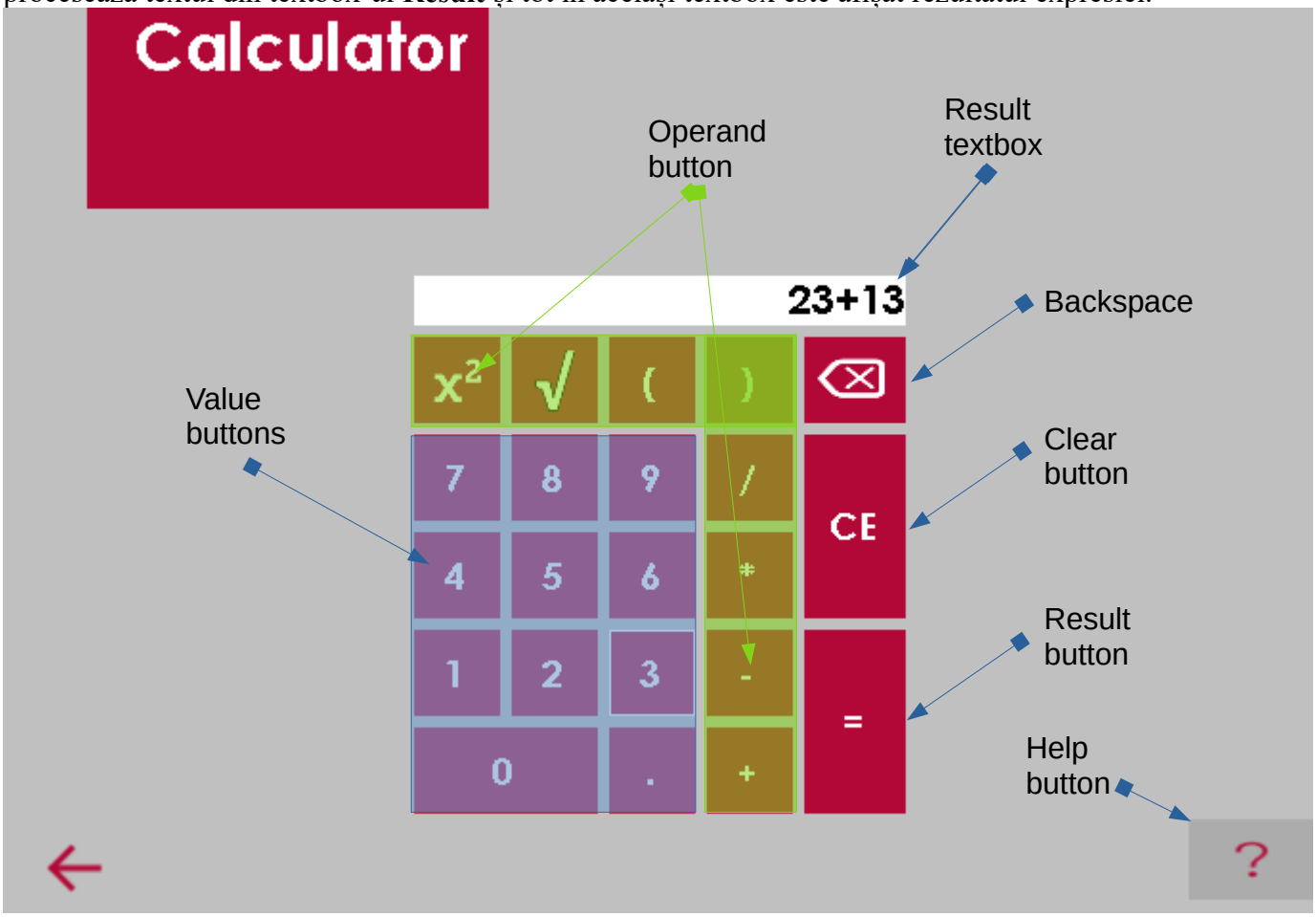
4. Prime



La apăsarea butonului **Calculate,** textul din textbox-ul **Value** este transformat într-o valoarea de tip **int**. Prin apelul funcție **prim()** se verifică dacă valoarea introdusă este sau nu un număr prim. În dependență de valoarea returnată label-ul **Result** afișează dacă valoarea data este un număr prim sau nu.

5. Calculator

La apăsarea butoanelor **Value** sau **Operand** în textbox-ul **Result** este adăugat la sfârșit textul corespunzător fiecărui buton apăsat. Butonul **Backspace** șterge ultimul caracter din textbox, butonul **Clear** șterge tot conținutul din textbox. La apăsarea butonului **Result**, cu ajutorul clasei **MathParser,** se procesează textul din textbox-ul **Result** și tot în același textbox este afișat rezultatul expresiei.

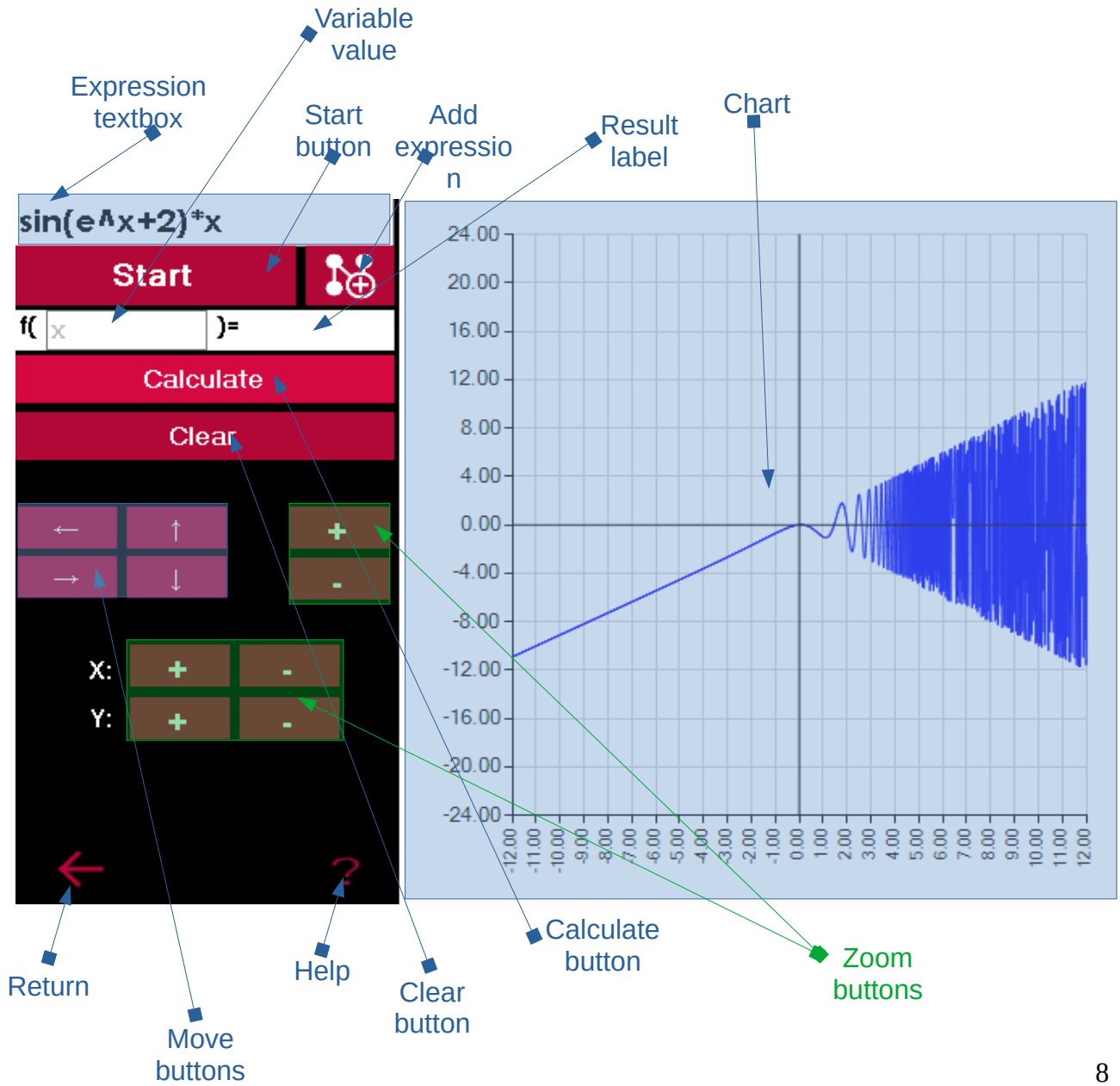


Fiecărui buton de tip **Value** sau **Operand** (excepție fiind parantezele, ridicarea la putere și radicalul) îi este definit evenimentul **buttonNr_Click(sender, e)** care adaogă la sfârșitul textului din textbox textul corespunzător fiecărui buton.

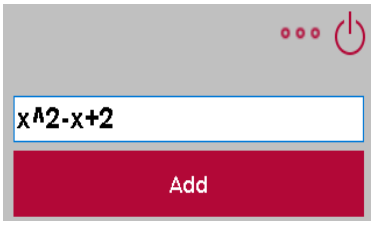
```
private void buttonNr_Click(object sender, EventArgs e)
        { if (textBox_Result.Text == "0.00")
            {
                textBox_Result.Text = "";
                textBox_Result.ForeColor = Color.Black;
            }
            if (textBox_Result.Text == result)
                textBox_Result.Text = "";
            Button button = (Button)sender;
            textBox_Result.Text = textBox_Result.Text + button.Text;}
```

6. Graph

Obiectul **graph1** este un **User Control**. La apăsarea butonului **Start** textul din textbox-ul **Expression** este analizat cu ajutorul clasei **MathParser** care returnează valoarea corespunzătoare expresiei pentru



fiecare variabilă **X** din intervalul **[xmin,xmax].** Valorile date sunt adăugate sub formă de puncte **(x,f(x))** în seria **First,** care este o serie corespunzătoare obiectului **chart1.** Seria dată este de tipul linie.
La apăsarea butonului **Calculate,** textul din textbox-ul **Expression** este procesat, primind ca variabilă **X**, valoarea introdusă în textbox-ul **Variable value**. Rezultatul returnat este afișat în label-ul **Result**.
Butonul **Clear** șterge toate seriile din graficul **chart1**.
Butoanele **Move** și **Zoom**, prin modificarea intervalelor definire pe axa **X** și **Y** realizează atât deplasarea pe axa absciselor și a ordonatelor cât și apropierea sau deplasarea a celor două axe.



Prin apăsarea butonului **Add expression** este inițializat un obiect de tip **Form** care prin apăsarea butonului **Add** apelează funcția **Add_Exp** a obiectului owner **(Chart)** care procesează textul din **textbox** și adaugă graficul corespunzător expresiei în obiectul **chart1**. Un algoritm similar este folosit și la adăugarea graficului inițial.

```
public void Add_Exp(string str)
        {
//Verifică dacă seria care urmează să fie introdusă nu este deja inițializată
            if (chart1.Series.IndexOf(str) != -1)
                chart1.Series[str].Points.Clear();
            else
                chart1.Series.Add(str);
//Definește seria dată cu proprietatea de tip linie
            chart1.Series[str].ChartType = SeriesChartType.Line;
            str = graphAdd.get_Expression();
            try
            {
                MathParser g = new MathParser();
                double y;
//Atribuie valori variabilei x în intervalul [xmin,xmax]
                for (double i = xmin; i <= xmax; i += 0.01F)
                {
                    g.setArgumentValue(i);
//Calculează valoarea expresiei
                    y = g.Calculate(str);
                    if (Math.Abs(y) > 99999 || Math.Abs(y) < -99999)
                    { }
                    else
                    {
                        if(y!=double.NaN)
//Adaugă punctele respective în serie
                        chart1.Series[str].Points.AddXY(i, y);
                    }
                }
            }
            catch{}
        }
```

Descrierea clasei MathParser

Clasa **MathParser** reprezintă o clasă cu ajutorul căreia expresiile matematice pot fi procesate și calculate valoarea sa numerică.
Una din proprietățile clasei date este posibilitatea de a defini o variabilă în cadrul unei expresii. Pentru a seta valoarea variabilei **x** este utilizată funcția **`setArgumentValue(double var),`** care primește ca parametru o valoarea de tip double și o salvează în cadrul obiectului sub forma de variabila **varX**.

```
1 public void setArgumentValue(double var)
2         {
3             varX = var;
4         }
```

Pentru evitarea excepțiilor produse de caracterele „*spațiu*” într-o expresie, este folosită funția **`public bool check(string str).`** Care verifică dacă expresia primită este sau nu validă.

```
1 public bool check(string str)
2         {
3             for (int i = 1; i < str.Length; i++)
4                 if (str[i] == ' ')
5                     return false;
6             return true;
7         }
```

Funcția principală a clasei o reprezintă funcția **`Calculate(string newExpr).`** Aceasta primește sub format de text expresia care trebuie calculată. Prin apeluri efective ale funcțiilor din cadrul acestei clase, funcția **Calculate()** verifică dacă valoarea returnabilă este validă și o returnează prin obiectul creat de tip **MathParser**.

```
1 public double Calculate(string newExpr)
2         {
3 //Transformă textul expresie într-o variabila de tip char-vector.
4             char[] expr = newExpr.ToCharArray();
5 //Calculează valoarea numerică a expresiei
6             double result = parseParanthesis(expr);
7 //Verifică dacă această valoarea este validă
8             if (result !=double.NaN)
9                 return result;
10            else
11                return double.NaN;
12        }
```

Pentru calculul valorii numerice a unei expresii în primul rând se verifică dacă în componența acestei expresii există paranteze. Dacă sunt găsite paranteze valide (care se deschid și se închid un număr egal

de ori) atunci se calculează recursiv valoarea expresiei dintre cele două paranteze până la dispariția acestor paranteze.

```
private double parseParanthesis(char[] expr)
        {
            int indexStart = 0;
            bool exParanthesis = false;
//Verifică dacă parantezele se dechid.
            while (indexStart < expr.Length - 1)
            {
                if (expr[indexStart] == '(')
                {
                    exParanthesis = true;
                    break;
                }
                indexStart++;

            }
            int indexEnd = indexStart + 1;
            int nr = 0;
//Verifică dacă parantezele se închid de același număr de ori de câte se deschid.
            while (indexEnd < expr.Length)
            {
                if (expr[indexEnd] == '(')
                    nr++;
                if (expr[indexEnd] == ')')
                {
                    if (nr == 0)

                    {
                        exParanthesis = true;
                        break;
                    }
                    else nr--;
                }
                indexEnd++;
            }
//Dacă au fost găsite paranteze se calculează valoarea expresiei dintre acestea.
            if (exParanthesis == true)
            {
//Se inițializează un nou obiect de tip MathParser cu ajutorul căreia se va calcula valoarea //
expresie dintre paranteze.
                MathParser mathParser = new MathParser();
                mathParser.setArgumentValue(varX);
//Expresia inițială este divizată în trei părți: până, între și după paranteze.
                string expression = new string(expr);
                string expression1 = expression;
                string expression2 = expression;
                expression = expression.Substring(0, indexEnd);
                expression = expression.Substring(indexStart + 1);
                if (indexStart - 1 >= 0)
                {
                    expression1 = expression1.Substring(0, indexStart);
                }
                else expression1 = "";
                if (indexEnd + 1 < expression2.Length)
                {
                    expression2 = expression2.Substring(indexEnd + 1);
                }
```

```
                else expression2 = "";
//Se calculează valoarea expresiei dintre paranteze.
                double result = mathParser.Calculate(expression);
                string expressionResult;
                if (result != double.NaN)
                    expressionResult = expression1 + result.ToString() + expression2;
                else return double.NaN;
//Expresia inițială este înlocuita cu echivalentul său fără paranteze.
                expr = expressionResult.ToCharArray();

                return parseParanthesis(expr);
            }
            else
//Dacă nu sunt găsite paranteze are loc calculul simplu.
                return parseSummands(expr, 0);

        }
```

Pentru adunările, scăderile, înmulțirile, împărțirile și ridicarea la putere sunt utilizate funcții simimilare celei de jos.

```
private double parseSummands(char[] expr, int index)
        {
//Calculează valoarea expresiei până la operatorul găsit (+,-,*,/,^).
            double x = parseFactors(expr, ref index);
//Caută operatorul (+,-,*,/,^).
            while (true)
            {
                char op = expr[index];
//Verifică dacă caracterul găsit este operator.
                if (op != '+' && op != '-')
                    return x;
                index++;
                double y = parseFactors(expr, ref index);
//Verifică tipul operatorului și calculează operația corespunzătoare dintre caracterul din
//stânga și dreapta operatorului.
                if (op == '+')
                    x += y;
                else
                    x -= y;
            }
        }
```

Pentru a se efectua operațiile dintre două numere este necesară o funcție care poate determina dacă caracterul dat este sau nu o valoarea numerică. Funcția **GetDouble(char[] expr, ref int index)** verifică dacă caracterul de pe poziția **index** este sau nu o valoare numerică, în cazul afirmativ aceasta continuă căutarea cu o poziție mai la stânga și salvează valoarea dată într-o variabile de tip **string**. De asemenea, în cazul în care este găsită o constantă deja definită sau variabila **X** funcștia **GetDouble()** returnează valoarea corespunzătoare constantei sau variabilei **X**.

```
private double GetDouble(char[] expr, ref int index)
        {
```

```
            string dbl = "";
            int indexInitial = index;
//Cât timp caracterul de pe poziția index este o cifră, constantă sau variabilă ciclul continuă
//până nu este găsit un operator sau un caracter necunoscut.
            while (((int)expr[index] >= '0' && (int)expr[index] <= '9') || expr[index] == '.'
|| expr[index] == '-' || expr[index] == 'e' || (expr[index] == 'p' && expr[index + 1] == 'i')
|| expr[index] == 'x')
            {
//Pentru returnarea valorilor negative, în cazul în care minusul nu este la începutul numărului
//se iese din ciclu și se returnează valoarea gastită.
                if (expr[index] == '-' && index != indexInitial)
                {
                    break;
                }
//Daca se găsesc două caracter -- acestea sunt sărită, pentru a fi echivalate cu +.
                if (expr[index] == '-' && expr[index + 1] == '-' && index == indexInitial)
                {
                    index += 2;
                }
//Dacă caracterul expr[index] este o constantă (e sau pi) atunci se returnează echivalentul
//numeric a acesteia.
                if (expr[index] == 'e')
                {
                    if (index + 1 < expr.Length)
                        index++;
                    return Math.E;
                }
                if (expr[index] == 'p' && expr[index + 1] == 'i')
                {
                    if (index + 2 < expr.Length)
                        index += 2;
                    return Math.PI;
                }
//Dacă expr[index] este este x atunci se returnează valoarea sa deja definită.
                if (expr[index] == 'x')
                {
                    if (index + 1 < expr.Length)
                        index++;
                    return varX;
                }
//Dacă caracterul expr[index] nu este niciunul din cazurile de mai sus (este o cifră sau punct)
atunci în șirul de caractere dbl este adăugat la sfârșit caracterul dat.
                else
                {
                    dbl = dbl + expr[index].ToString();
                    index++;
                }

//Dacă ciclul ajunge la sfârșitul șirului expr se scade unu din index și se iese din ciclu
                if (index == expr.Length)
                {
                    index--;
                    break;
                }
            }
//Dacă nu s-a găsit nici un caracter returnabil se returnează valoarea Nan.
            if (dbl == "")
                return double.NaN;
            else
//Dacă sirul dbl nu este vid se returnează echivalentul său numeric.
```

```
64                  return double.Parse(dbl);
65          }
```

Posibilități de dezvoltare a aplicației

Aplicația dată reprezintă o platformă predispusă modificărilor și adăugării noilor posibilități. De exemplu poate fi introdusă posibilitatea de analiză a obiectelor tridimensionale (suprafața, volumul etc.), calculul matricial (înmulțirea, adunarea și ridicarea la putere a matricilor) și a multor alte aspecte legate de matematică.
De asemenea, pentru a crea o interfață mai prietenoasă utilizatorului final este necesară crearea unei aplicații **Tutorial** care ar putea introduce utilizatorul în funcționalitatea aplicație.
O altă posibilitatea de dezvoltare ar fi completarea clasei **MathParser** cu noi funcții și constante. De asemenea este necesară și optimizarea lucrului acesteia pentru a evita excepții și greșelile de calcul.

Webografie

1. http://mathparser.org/
2. https://www.mathopenref.com/coordpolygonarea.html
3. https://www.c-sharpcorner.com/article/generating-random-number-and-string-in-C-Sharp/
4. https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.datavisualization.charting.chart?view=netframework-4.8