|  |  |  |
| --- | --- | --- |
| Министерство науки и высшего образования  Российской Федерации | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования | | |
| «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра прикладной математики | | |
|  | | |
| Практическое задание № 3 | | |
| по дисциплине «объектно-ориентированное программирование с | | |
| использованием С++/C#» | | |
|  | | |
|  | | |
|  | Факультет: | ПМИ |
| Группа: | ПМ-71 |
| Студент: | Востриков Вячеслав |
| Преподаватель: | Ступаков Илья Михайлович |
|  |  |
|  |  |
|  |  |
|  |  |
|  | | |
| Новосибирск | | |
| 2019 | | |

1. Условие задачи

Сделать иерархию классов для вычисления математических выражений.

1. Базовый абстрактный класс

abstract class Expr

{

   public abstract double Compute(IReadOnlyDictionary<string, double> variableValues);

   public abstract IEnumerable<string> Variables { get; protected set;}

   public abstract bool IsConstant { get; }

   public abstract bool IsPolynom { get; }

}

1. Абстрактные классы

UnaryOperation, BinaryOperation, Function

1. Классы реализующие арифметические операции и класс Variable, Constant.
2. Сделать для этих классов перегрузку операторов.
3. Классы реализующие функции
   1. Степенные
   2. Тригонометрические
   3. Обратные тригонометрические
   4. Гиперболические
   5. Обратные гиперболические
4. Доп. задания
   1. Дифференцирование (3)
   2. Разбор выражений (4)
   3. Упрощение выражений (4)
   4. Интегрирование (4)
   5. Векторная арифметика (3)
   6. Добавление своих функций (2)

**Пример**

var a = new Variable("a");

var b = new Variable("b");

var expr0 = new Mult(new Add(a, b), new SinFunc(new Divide(a, new Constant(2))));

var expr = (a + b) \* Sin(a / 2);

Console.WriteLine(expr);

Console.WriteLine(expr.Compute(new Dictionary<string, double>{["a"] = 5, ["b"] = 3});

1. Код программы

using System;

using System.Collections.Generic;

using System.Linq;

namespace Laba3

{

using static Functions;

public abstract class Expr

{

public abstract double Compute(IReadOnlyDictionary<string, double> variableValues);

public abstract IEnumerable<string> Variables { get; }

public abstract bool IsConstant { get; }

public abstract bool IsPolynom { get; }

public static Add operator +(Expr a, Expr b) => new Add(a, b);

public static UnaryMin operator -(Expr c) => new UnaryMin(c);

public static Sub operator -(Expr a, Expr b) => new Sub(a, b);

public static Mult operator \*(Expr a, Expr b) => new Mult(a, b);

public static Div operator /(Expr a, Expr b) => new Div(a, b);

public static implicit operator Expr(double x) => new Constant(x);

}

public static class Functions

{

public static ArcSin ArcSin(Expr a) => new ArcSin(a);

public static ArcCos ArcCos(Expr a) => new ArcCos(a);

public static ArcTg ArcTg(Expr a) => new ArcTg(a);

public static ArcCtg ArcCtg(Expr a) => new ArcCtg(a);

public static Cos Cos(Expr a) => new Cos(a);

public static Sin Sin(Expr a) => new Sin(a);

public static Ctg Ctg(Expr a) => new Ctg(a);

public static Tg Tg(Expr a) => new Tg(a);

public static Sqrt Sqrt(Expr a) => new Sqrt(a);

public static Sinh Sinh(Expr a) => new Sinh(a);

public static Cosh Cosh(Expr a) => new Cosh(a);

public static Tanh Tanh(Expr a) => new Tanh(a);

public static CTanh CTanh(Expr a) => new CTanh(a);

}

public class Constant : Expr

{

public double Value { get; }

public override IEnumerable<string> Variables => Enumerable.Empty<string>();

public override bool IsConstant => true;

public override bool IsPolynom => true;

public Constant(double value) => this.Value = value;

public override double Compute(IReadOnlyDictionary<string, double> variableValues) => Value;

public override string ToString()

{

return $"{Value}";

}

}

public class Variable : Expr

{

public string Name { get; }

public Variable(string name) => this.Name = name;

public override IEnumerable<string> Variables => new List<string> { Name };

public override bool IsConstant => false;

public override bool IsPolynom => true;

public override double Compute(IReadOnlyDictionary<string, double> variableValues)

{

if (variableValues.TryGetValue(Name, out double value))

return value;

else throw new ArgumentException("Not found variable");

}

public override string ToString()

{

return $"{Name}";

}

}

abstract public class BinaryOperation : Expr

{

protected Expr Arg1 { get; set; }

protected Expr Arg2 { get; set; }

public BinaryOperation(Expr arg1, Expr arg2)

{

this.Arg1 = arg1;

this.Arg2 = arg2;

}

public override IEnumerable<string> Variables => Arg1.Variables.Union<string>(Arg2.Variables);

public override bool IsConstant => Arg1.IsConstant && Arg2.IsConstant;

}

public class Add : BinaryOperation

{

public Add(Expr arg1, Expr arg2) : base(arg1, arg2) { }

public override double Compute(IReadOnlyDictionary<string, double> variableValues)

{

return Arg1.Compute(variableValues) + Arg2.Compute(variableValues);

}

public override bool IsPolynom => Arg1.IsPolynom && Arg2.IsPolynom;

public override string ToString()

{

return $"({Arg1}+{Arg2})";

}

}

public class Sub : BinaryOperation

{

public Sub(Expr arg1, Expr arg2) : base(arg1, arg2) { }

public override double Compute(IReadOnlyDictionary<string, double> variableValues) => Arg1.Compute(variableValues) - Arg2.Compute(variableValues);

public override bool IsPolynom => Arg1.IsPolynom && Arg2.IsPolynom;

public override string ToString()

{

return $"({Arg1}-{Arg2})";

}

}

public class Mult : BinaryOperation

{

public Mult(Expr arg1, Expr arg2) : base(arg1, arg2) { }

public override double Compute(IReadOnlyDictionary<string, double> variableValues) => Arg1.Compute(variableValues) \* Arg2.Compute(variableValues);

public override bool IsPolynom => Arg1.IsPolynom && Arg2.IsPolynom;

public override string ToString()

{

return ($"({Arg1}\*{Arg2})");

}

}

public class Div : BinaryOperation

{

public Div(Expr arg1, Expr arg2) : base(arg1, arg2) { }

public override double Compute(IReadOnlyDictionary<string, double> variableValues)

{

if (Arg2.Compute(variableValues) == 0)

throw new ArgumentException("Знаменатель равен 0!");

else

return Arg1.Compute(variableValues) / Arg2.Compute(variableValues);

}

public override bool IsPolynom => Arg1.IsPolynom && Arg2.IsConstant;

public override string ToString()

{

return $"({Arg1}/{Arg2})";

}

}

public abstract class UnaryOperations : Expr

{

protected Expr Arg1 { get; set; }

public UnaryOperations(Expr arg1)

{

this.Arg1 = arg1;

}

public override IEnumerable<string> Variables => Arg1.Variables;

public override bool IsConstant => Arg1.IsConstant;

public override bool IsPolynom => Arg1.IsPolynom;

}

public class UnaryMin : UnaryOperations

{

public UnaryMin(Expr arg1) : base(arg1) { }

public override double Compute(IReadOnlyDictionary<string, double> variableValues) => -Arg1.Compute(variableValues);

public override string ToString()

{

return $"(-{Arg1})";

}

}

public abstract class Function : Expr

{

protected Expr Arg1 { get; set; }

public Function(Expr arg1)

{

this.Arg1 = arg1;

}

public override IEnumerable<string> Variables => Arg1.Variables;

}

public class Sqrt : Function

{

public Sqrt(Expr arg1) : base(arg1) { }

public override double Compute(IReadOnlyDictionary<string, double> variableValues)

{

double sup = Arg1.Compute(variableValues);

if (sup >= 0)

return Math.Sqrt(sup);

else throw new ArgumentException("Отрицательное число под корнем!");

}

public override string ToString()

{

return $"Sqrt({Arg1})";

}

public override bool IsConstant => Arg1.IsConstant;

public override bool IsPolynom => Arg1.IsConstant;

}

public class ArcSin : Function

{

public ArcSin(Expr arg1) : base(arg1) { }

public override double Compute(IReadOnlyDictionary<string, double> variableValues)

{

double sup = Arg1.Compute(variableValues);

if (sup > -1 && sup < 1)

return Math.Asin(sup);

else throw new ArgumentException("Нет решения!");

}

public override string ToString()

{

return $"ArcSin({Arg1})";

}

public override bool IsConstant => Arg1.IsConstant;

public override bool IsPolynom => Arg1.IsConstant;

}

public class ArcCos : Function

{

public ArcCos(Expr arg1) : base(arg1) { }

public override double Compute(IReadOnlyDictionary<string, double> variableValues)

{

double sup = Arg1.Compute(variableValues);

if (sup > -1 && sup < 1)

return Math.Acos(sup);

else throw new ArgumentException("Нет решения!");

}

public override string ToString()

{

return $"ArcCos({Arg1})";

}

public override bool IsConstant => Arg1.IsConstant;

public override bool IsPolynom => Arg1.IsConstant;

}

public class ArcTg : Function

{

public ArcTg(Expr arg1) : base(arg1) { }

public override double Compute(IReadOnlyDictionary<string, double> variableValues) => Math.Atan(Arg1.Compute(variableValues));

public override string ToString()

{

return $"ArcTg({Arg1})";

}

public override bool IsConstant => Arg1.IsConstant;

public override bool IsPolynom => Arg1.IsConstant;

}

public class ArcCtg : Function

{

public ArcCtg(Expr arg1) : base(arg1) { }

public override double Compute(IReadOnlyDictionary<string, double> variableValues) => Math.PI / 2 - ArcTg(Arg1).Compute(variableValues);

public override string ToString()

{

return $"ArcCtg({Arg1})";

}

public override bool IsConstant => Arg1.IsConstant;

public override bool IsPolynom => Arg1.IsConstant;

}

public class Sin : Function

{

public Sin(Expr arg1) : base(arg1) { }

public override double Compute(IReadOnlyDictionary<string, double> variableValues) => Math.Sin(Arg1.Compute(variableValues));

public override string ToString()

{

return $"Sin({Arg1:f1})";

}

public override bool IsConstant => Arg1.IsConstant;

public override bool IsPolynom => Arg1.IsConstant;

}

public class Cos : Function

{

public Cos(Expr arg1) : base(arg1) { }

public override double Compute(IReadOnlyDictionary<string, double> variableValues) => Math.Cos(Arg1.Compute(variableValues));

public override string ToString()

{

return $"Cos({Arg1:f1})";

}

public override bool IsConstant => Arg1.IsConstant;

public override bool IsPolynom => Arg1.IsConstant;

}

public class Tg : Function

{

public Tg(Expr arg1) : base(arg1) { }

public override double Compute(IReadOnlyDictionary<string, double> variableValues) => Math.Tan(Arg1.Compute(variableValues));

public override string ToString()

{

return $"Tg({Arg1:f1})";

}

public override bool IsConstant => Arg1.IsConstant;

public override bool IsPolynom => Arg1.IsConstant;

}

public class Ctg : Function

{

public Ctg(Expr arg1) : base(arg1) { }

public override double Compute(IReadOnlyDictionary<string, double> variableValues) => 1 / Tg(Arg1).Compute(variableValues);

public override string ToString()

{

return $"Ctg({Arg1:f1})";

}

public override bool IsConstant => Arg1.IsConstant;

public override bool IsPolynom => Arg1.IsConstant;

}

public class Sinh : Function

{

public Sinh(Expr arg1) : base(arg1) { }

public override bool IsConstant => Arg1.IsConstant;

public override bool IsPolynom => Arg1.IsConstant;

public override double Compute(IReadOnlyDictionary<string, double> variableValues) => Math.Sinh(Arg1.Compute(variableValues));

public override string ToString()

{

return $"Sinh({Arg1:f1})";

}

}

public class Cosh : Function

{

public Cosh(Expr arg1) : base(arg1) { }

public override bool IsConstant => Arg1.IsConstant;

public override bool IsPolynom => Arg1.IsConstant;

public override double Compute(IReadOnlyDictionary<string, double> variableValues) => Math.Cosh(Arg1.Compute(variableValues));

public override string ToString()

{

return $"Cosh({Arg1:f1})";

}

}

public class Tanh : Function

{

public Tanh(Expr arg1) : base(arg1) { }

public override double Compute(IReadOnlyDictionary<string, double> variableValues) => Math.Tanh(Arg1.Compute(variableValues));

public override string ToString()

{

return $"Tanh({Arg1:f1})";

}

public override bool IsConstant => Arg1.IsConstant;

public override bool IsPolynom => Arg1.IsConstant;

}

public class CTanh : Function

{

public CTanh(Expr arg1) : base(arg1) { }

public override double Compute(IReadOnlyDictionary<string, double> variableValues)

{

double sup = Arg1.Compute(variableValues);

if (sup != 0)

return 1 / Math.Tanh(sup);

else throw new ArgumentException("0 в знаменателе!");

}

public override string ToString()

{

return $"CTanh({Arg1:f1})";

}

public override bool IsConstant => Arg1.IsConstant;

public override bool IsPolynom => Arg1.IsConstant;

}

class Program

{

static void Main(string[] args)

{

var a = new Variable("a");

var b = new Variable("b");

var c = new Variable("c");

Expr expr = Sqrt(9) + a + b + Sin(c) + Cos(c);

Console.WriteLine(" ");

Console.WriteLine(expr);

Console.WriteLine("Выражение постоянно? {0}", expr.IsConstant);

Console.WriteLine($"Выражение полином? {expr.IsPolynom}");

Console.WriteLine($" = { expr.Compute(new Dictionary<string, double> { ["a"] = 5, ["b"] = 2.5, ["c"] = 0 })}");

var h = new Variable("h");

Expr expr1 = Tanh(h) \* CTanh(h);

Console.WriteLine(" ");

Console.WriteLine(expr1);

Console.WriteLine("Выражение постоянно? {0}", expr1.IsConstant);

Console.WriteLine($"Выражение полином? {expr1.IsPolynom}");

Console.WriteLine($" = { expr1.Compute(new Dictionary<string, double> { ["h"] = 2 })}");

Expr expr2 = a + b;

Console.WriteLine(" ");

Console.WriteLine(expr2);

Console.WriteLine("Выражение постоянно? {0}", expr2.IsConstant);

Console.WriteLine($"Выражение полином? {expr2.IsPolynom}");

Console.WriteLine($" = { expr2.Compute(new Dictionary<string, double> { ["a"] = 2, ["b"] = 1 })}");

var z = new Constant(2);

var g = new Constant(2);

Expr expr3 = z + g;

Console.WriteLine(" ");

Console.WriteLine(expr3);

Console.WriteLine("Выражение постоянно? {0}", expr3.IsConstant);

Console.WriteLine($"Выражение полином? {expr3.IsPolynom}");

Console.WriteLine($" = { expr3.Compute(new Dictionary<string, double> { })}");

Console.ReadKey();

}

}

}

1. Результаты работы программы

