

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FunctionProgramming

{

static class Expressions

{

public static IEnumerable<T> TransformCollection<T>(this IEnumerable<T> col) where T:struct

{

**Func**<T, T> func = x => (dynamic)x/2;

ICollection<T> tmp = new Collection<T>();

for (int i = 0; i < col.Count()-1; i++)

{

T result = default (T);

result=func(col.ElementAt(i));

if(i%2!=0)

result +=(dynamic) new **Func**<T>(() => i != col.Count()-1 ? col.ElementAt(i+1):col.ElementAt(i-1))();

tmp.Add(result);

}

return tmp;

}

}

class Program

{

static void Main(string[] args)

{

double []arr = {1.2,23.4,11,44.8,23,10,12.5};

var t = (Array.ConvertAll(arr,new Converter<double,int>((x)=>(int)x))).TransformCollection();

foreach (var d in t)

{

Console.WriteLine(d);

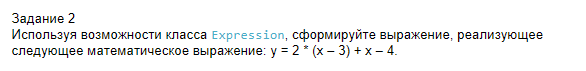
}

Console.ReadKey();

}

}

}



using System;

using System.Collections.Generic;

using System.Linq;

using System.Linq.Expressions;

using System.Text;

using System.Threading.Tasks;

namespace MyExpression

{

class Program

{

static void Main(string[] args)

{

MyExpr(10);

Console.ReadKey();

}

static private void MyExpr<T>(T x)

{

ParameterExpression p1 = Expression<T>.Parameter(typeof (T), "x");

ConstantExpression pc1 = Expression<T>.Constant(2);

ConstantExpression pc2 = Expression<T>.Constant(3);

ConstantExpression pc3 = Expression<T>.Constant(4);

BinaryExpression be = Expression.Subtract(p1, pc2);

BinaryExpression be1 = Expression.Multiply(pc1, be);

BinaryExpression be2 = Expression.Add(be1, p1);

BinaryExpression be3 = Expression.Subtract(be2, pc3);

Expression<**Func**<T, T>> expr = Expression.Lambda<**Func**<T,T>>(be3,p1);

Console.Write(expr.Body+" = ");

Console.WriteLine(expr.Compile().Invoke(x));

}

}

}