1.1 ./Platform.RegularExpressions.Transformer.CSharpToCpp/CSharpToCppTransformer.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text.RegularExpressions;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.RegularExpressions.Transformer.CSharpToCpp
{
    public class CSharpToCppTransformer : Transformer
    {
        public static readonly IList<ISubstitutionRule> FirstStage = new List<SubstitutionRule>
        {
            // // ...
            //
            (new Regex(@"(\r?\n)?[ \t]+//+.+"), "", null, 0),
            // #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
            //
            (new Regex(@"^\s*?\#pragma[\sa-zA-Z0-9]+$"), "", null, 0),
            // {\n\n\n
            // {
            (new Regex(@"{\s+[\r\n]+"), "{" + Environment.NewLine, null, 0),
            // Platform.Collections.Methods.Lists
            // Platform::Collections::Methods::Lists
            (new Regex(@"(namespace[^\r\n]+?)\.([^\r\n]+?)"), "$1::$2", null, 20),
            // out TProduct
            // TProduct
            (new Regex(@"(?<before>(<|, ))(in|out) (?<typeParameter>[a-zA-Z0-9]+)(?<after>(>|,))"), "${before}${typeParameter}${after}", null, 10),
            // public ...
            // public: ...
            (new Regex(@"(?<newLineAndIndent>\r?\n?[ \t]*)(?<before>[^\{\(\r\n]*)(?<access>private|protected|public)[ \t]+(?![^\{\(\r\n]*(interface|class|struct)[^\{\(\r\n]*[\{\(\r\n])"), "${newLineAndIndent}${access}: ${before}", null, 0),
            // public: static bool CollectExceptions { get; set; }
            // public: static bool CollectExceptions;
            (new Regex(@"(?<before>(private|protected|public): (static )?[^\r\n]+ )(?<name>[a-zA-Z0-9]+) {[^;}]*(?<=\W)get;[^;}]*(?<=\W)set;[^;}]*}"), "${before}${name};", null, 0),
            // public abstract class
            // class
            (new Regex(@"((public|protected|private|internal|abstract|static) )*(?<category>interface|class|struct)"), "${category}", null, 0),
            // class GenericCollectionMethodsBase<TElement> {
            // template <typename TElement> class GenericCollectionMethodsBase {
            (new Regex(@"class ([a-zA-Z0-9]+)<([a-zA-Z0-9]+)>([^{]+){"), "template <typename $2> class $1$3{", null, 0),
            // static void TestMultipleCreationsAndDeletions<TElement>(SizedBinaryTreeMethodsBase<TElement> tree, TElement* root)
            // template<typename T> static void TestMultipleCreationsAndDeletions<TElement>(SizedBinaryTreeMethodsBase<TElement> tree, TElement* root)
            (new Regex(@"static ([a-zA-Z0-9]+) ([a-zA-Z0-9]+)<([a-zA-Z0-9]+)>\(([^\)\r\n]+)\)"), "template <typename $3> static $1 $2($4)", null, 0),
            // interface IFactory<out TProduct> {
            // template <typename TProduct> class IFactory { public:
            (new Regex(@"interface (?<interface>[a-zA-Z0-9]+)<(?<typeParameters>[a-zA-Z0-9,]+)>(?<whitespace>[^{]+){"), "template <typename...> class ${interface}; template <typename ${typeParameters}> class ${interface}<${typeParameters}>${whitespace}{" + Environment.NewLine + " public:", null, 0),
            // template <typename TObject, TProperty, TValue>
            // template <typename TObject, typename TProperty, TValue>
            (new Regex(@"(?<before>template <(, )?typename [a-zA-Z0-9]+)+, )(?<typeParameter>[a-zA-Z0-9]+)(?<after>(,|>))"), "${before}typename ${typeParameter}${after}", null, 10),
            // Insert markers
            // private: static void BuildExceptionString(this StringBuilder sb, Exception exception, int level)
            // /*~extensionMethod~BuildExceptionString~*/private: static void BuildExceptionString(this StringBuilder sb, Exception exception, int level)
```

```
            (new Regex(@"private: static [^\r\n]+ (?<name>[a-zA-Z0-9]+)\(this [^\)\r\n]+\)"),
            ↪  "/*~extensionMethod~${name}~*/$0", null, 0),
            // Move all markers to the beginning of the file.
            (new Regex(@"\A(?<before>[^\r\n]+\r?\n(.|\n)+)(?<marker>/\*~extensionMethod~(?<name>
            ↪  [a-zA-Z0-9]+)~\*/)"), "${marker}${before}", null,
            ↪  10),
            // /*~extensionMethod~BuildExceptionString~*/...sb.BuildExceptionString(exception.In
            ↪  nerException, level +
            ↪  1);
            // /*~extensionMethod~BuildExceptionString~*/...BuildExceptionString(sb,
            ↪  exception.InnerException, level + 1);
            (new Regex(@"(?<before>/\*~extensionMethod~(?<name>[a-zA-Z0-9]+)~\*/(.|\n)+\W)(?<var
            ↪  iable>[_a-zA-Z0-9]+)\.\k<name>\("), "${before}${name}(${variable}, ", null,
            ↪  50),
            // Remove markers
            // /*~extensionMethod~BuildExceptionString~*/
            //
            (new Regex(@"/\*~extensionMethod~[a-zA-Z0-9]+~\*/"), "", null, 0),
            // (this
            // (
            (new Regex(@"\(this "), "(", null, 0),
            // public: static readonly EnsureAlwaysExtensionRoot Always = new
            ↪  EnsureAlwaysExtensionRoot();
            // public:inline static EnsureAlwaysExtensionRoot Always;
            (new Regex(@"(?<access>(private|protected|public): )?static readonly
            ↪  (?<type>[a-zA-Z0-9]+) (?<name>[a-zA-Z0-9_]+) = new \k<type>\(\);"),
            ↪  "${access}inline static ${type} ${name};", null, 0),
            // public: static readonly string ExceptionContentsSeparator = "---";
            // public: inline static const char* ExceptionContentsSeparator = "---";
            (new Regex(@"(?<access>(private|protected|public): )?static readonly string
            ↪  (?<name>[a-zA-Z0-9_]+) = ""(?<string>(\""|[^""\r\n])+)"";"), "${access}inline
            ↪  static const char* ${name} = \"${string}\";", null, 0),
            // private: const int MaxPath = 92;
            // private: static const int MaxPath = 92;
            (new Regex(@"(?<access>(private|protected|public): )?(const|static readonly)
            ↪  (?<type>[a-zA-Z0-9]+) (?<name>[_a-zA-Z0-9]+) = (?<value>[^;\r\n]+);"),
            ↪  "${access}static const ${type} ${name} = ${value};", null, 0),
            //  ArgumentNotNull(EnsureAlwaysExtensionRoot root, TArgument argument) where
            ↪  TArgument : class
            //  ArgumentNotNull(EnsureAlwaysExtensionRoot root, TArgument* argument)
            (new Regex(@"(?<before> [a-zA-Z]+\(([a-zA-Z *,]+, |))(?<type>[a-zA-Z]+)(?<after>(|
            ↪  [a-zA-Z *,]+)\))[ \r\n]+where \k<type> : class"), "${before}${type}*${after}",
            ↪  null, 0),
            // protected: abstract TElement GetFirst();
            // protected: virtual TElement GetFirst() = 0;
            (new Regex(@"(?<access>(private|protected|public): )?abstract
            ↪  (?<method>[^;\r\n]+);"), "${access}virtual ${method} = 0;", null, 0),
            // TElement GetFirst();
            // virtual TElement GetFirst() = 0;
            (new Regex(@"([\r\n]+[ ]+)((?!return)[a-zA-Z0-9]+ [a-zA-Z0-9]+\([^\)\r\n]*\))(;[
            ↪  ]*[\r\n]+)"), "$1virtual $2 = 0$3", null, 1),
            // protected: readonly TreeElement[] _elements;
            // protected: TreeElement _elements[N];
            (new Regex(@"(?<access>(private|protected|public): )?readonly
            ↪  (?<type>[a-zA-Z<>0-9]+)([\[\]]+) (?<name>[_a-zA-Z0-9]+);"), "${access}${type}
            ↪  ${name}[N];", null, 0),
            // protected: readonly TElement Zero;
            // protected: TElement Zero;
            (new Regex(@"(?<access>(private|protected|public): )?readonly
            ↪  (?<type>[a-zA-Z<>0-9]+) (?<name>[_a-zA-Z0-9]+);"), "${access}${type} ${name};",
            ↪  null, 0),
            // internal
            //
            (new Regex(@"(\W)internal\s+"), "$1", null, 0),
            // static void NotImplementedException(ThrowExtensionRoot root) => throw new
            ↪  NotImplementedException();
            // static void NotImplementedException(ThrowExtensionRoot root) { return throw new
            ↪  NotImplementedException(); }
            (new Regex(@"(^\s+)(private|protected|public)?(: )?(template \<[^>\r\n]+\> )?(static
            ↪  )?(override )?([a-zA-Z0-9]+
            ↪  )([a-zA-Z0-9]+)\(([^\(\r\n]*)\)\s+=>\s+throw([^;\r\n]+);"),
            ↪  "$1$2$3$4$5$6$7$8($9) { throw$10; }", null, 0),
            // SizeBalancedTree(int capacity) => a = b;
            // SizeBalancedTree(int capacity) { a = b; }
```

```
98              (new Regex(@"(^\s+)(private|protected|public)?(: )?(template \<[^>\r\n]+\> )?(static
                ↪  )?(override )?(void )?([a-zA-Z0-9]+)\(([^\(\r\n]*)\)\s+=>\s+([^;\r\n]+);"),
                ↪  "$1$2$3$4$5$6$7$8($9) { $10; }", null, 0),
99              // int SizeBalancedTree(int capacity) => a;
100             // int SizeBalancedTree(int capacity) { return a; }
101             (new Regex(@"(^\s+)(private|protected|public)?(: )?(template \<[^>\r\n]+\> )?(static
                ↪  )?(override )?([a-zA-Z0-9]+
                ↪  )([a-zA-Z0-9]+)\(([^\(\r\n]*)\)\s+=>\s+([^;\r\n]+);"), "$1$2$3$4$5$6$7$8($9) {
                ↪  return $10; }", null, 0),
102             // () => Integer<TElement>.Zero,
103             // () { return Integer<TElement>.Zero; },
104             (new Regex(@"\(\)\s+=>\s+([^,;\r\n]+?),"), "() { return $1; },", null, 0),
105             // => Integer<TElement>.Zero;
106             // { return Integer<TElement>.Zero; }
107             (new Regex(@"\)\s+=>\s+([^;\r\n]+?);"), ") { return $1; }", null, 0),
108             // () { return avlTree.Count; }
109             // [&]()-> auto { return avlTree.Count; }
110             (new Regex(@", \(\) { return ([^;\r\n]+); }"), ", [&]()-> auto { return $1; }",
                ↪  null, 0),
111             // Count => GetSizeOrZero(Root);
112             // GetCount() { return GetSizeOrZero(Root); }
113             (new Regex(@"(\W)([A-Z][a-zA-Z]+)\s+=>\s+([^;\r\n]+);"), "$1Get$2() { return $3; }",
                ↪  null, 0),
114             // Func<TElement> treeCount
115             // std::function<TElement()> treeCount
116             (new Regex(@"Func<([a-zA-Z0-9]+)> ([a-zA-Z0-9]+)"), "std::function<$1()> $2", null,
                ↪  0),
117             // Action<TElement> free
118             // std::function<void(TElement)> free
119             (new Regex(@"Action<([a-zA-Z0-9]+)> ([a-zA-Z0-9]+)"), "std::function<void($1)> $2",
                ↪  null, 0),
120             // Predicate<TArgument> predicate
121             // std::function<bool(TArgument)> predicate
122             (new Regex(@"Predicate<([a-zA-Z0-9]+)> ([a-zA-Z0-9]+)"), "std::function<bool($1)>
                ↪  $2", null, 0),
123             // var
124             // auto
125             (new Regex(@"(\W)var(\W)"), "$1auto$2", null, 0),
126             // unchecked
127             //
128             (new Regex(@"[\r\n]{2}\s*?unchecked\s*?$"), "", null, 0),
129             // throw new InvalidOperationException
130             // throw std::runtime_error
131             (new Regex(@"throw new (InvalidOperationException|Exception)"), "throw
                ↪  std::runtime_error", null, 0),
132             // void RaiseExceptionIgnoredEvent(Exception exception)
133             // void RaiseExceptionIgnoredEvent(const std::exception& exception)
134             (new Regex(@"(\(|, )(System\.Exception|Exception)( |\))"), "$1const
                ↪  std::exception&$3", null, 0),
135             // EventHandler<Exception>
136             // EventHandler<std::exception>
137             (new Regex(@"(\W)(System\.Exception|Exception)(\W)"), "$1std::exception$3", null, 0),
138             // override void PrintNode(TElement node, StringBuilder sb, int level)
139             // void PrintNode(TElement node, StringBuilder sb, int level) override
140             (new Regex(@"override ([a-zA-Z0-9 \*\+]+)(\([^\)\r\n]+?\))"), "$1$2 override", null,
                ↪  0),
141             // string
142             // const char*
143             (new Regex(@"(\W)string(\W)"), "$1const char*$2", null, 0),
144             // sbyte
145             // std::int8_t
146             (new Regex(@"(\W)sbyte(\W)"), "$1std::int8_t$2", null, 0),
147             // uint
148             // std::uint32_t
149             (new Regex(@"(\W)uint(\W)"), "$1std::uint32_t$2", null, 0),
150             // char*[] args
151             // char* args[]
152             (new Regex(@"([_a-zA-Z0-9:\*]?)\[\] ([a-zA-Z0-9]+)"), "$1 $2[]", null, 0),
153             // @object
154             // object
155             (new Regex(@"@([_a-zA-Z0-9]+)"), "$1", null, 0),
156             // using Platform.Numbers;
157             //
158             (new Regex(@"([\r\n]{2}|^)\s*?using [\.a-zA-Z0-9]+;\s*?$"), "", null, 0),
159             // struct TreeElement { }
160             // struct TreeElement { };
```

```
            (new Regex(@"(struct|class) ([a-zA-Z0-9]+)(\s+){([\sa-zA-Z0-9;:_]+?)}([^;])"), "$1 $2$3{$4};$5", null, 0),
            // class Program { }
            // class Program { };
            (new Regex(@"(struct|class) ([a-zA-Z0-9]+[^\r\n]*)([\r\n]+(?<indentLevel>[\t ]*)?)\{([\S\s]+?[\r\n]+\k<indentLevel>)\}([^;]|$)"), "$1 $2$3{$4};$5", null, 0),
            // class SizedBinaryTreeMethodsBase : GenericCollectionMethodsBase
            // class SizedBinaryTreeMethodsBase : public GenericCollectionMethodsBase
            (new Regex(@"class ([a-zA-Z0-9]+) : ([a-zA-Z0-9]+)"), "class $1 : public $2", null, 0),
            // class IProperty : ISetter<TValue, TObject>, IProvider<TValue, TObject>
            // class IProperty : public ISetter<TValue, TObject>, IProvider<TValue, TObject>
            (new Regex(@"(?<before>class [a-zA-Z0-9]+ : ((public [a-zA-Z0-9]+(<[a-zA-Z0-9 ,]+>)?, )+)?)(?<inheritedType>(?!public)[a-zA-Z0-9]+(<[a-zA-Z0-9 ,]+>)?)(?<after>(, [a-zA-Z0-9]+(?!>)|[ \r\n]+))"), "${before}public ${inheritedType}${after}", null, 10),
            // Insert scope borders.
            // ref TElement root
            // ~!root!~ref TElement root
            (new Regex(@"(?<definition>(?<= |\()(ref [a-zA-Z0-9]+|[a-zA-Z0-9]+(?<!ref)) (?<variable>[a-zA-Z0-9]+)(?=\)|, | =))"), "~!${variable}!~${definition}", null, 0),
            // Inside the scope of ~!root!~ replace:
            // root
            // *root
            (new Regex(@"(?<definition>~!(?<pointer>[a-zA-Z0-9]+)!~ref [a-zA-Z0-9]+ \k<pointer>(?=\)|, | =))(?<before>((?<!~!\k<pointer>!~)(.|\n))*?)(?<prefix>(\W|\())\k<pointer>(?<suffix>( |\)|;|,))"),
            "${definition}${before}${prefix}*${pointer}${suffix}", null, 70),
            // Remove scope borders.
            // ~!root!~
            //
            (new Regex(@"~!(?<pointer>[a-zA-Z0-9]+)!~"), "", null, 5),
            // ref auto root = ref
            // ref auto root =
            (new Regex(@"ref ([a-zA-Z0-9]+) ([a-zA-Z0-9]+) = ref(\W)"), "$1* $2 =$3", null, 0),
            // *root = ref left;
            // root = left;
            (new Regex(@"\*([a-zA-Z0-9]+) = ref ([a-zA-Z0-9]+)(\W)"), "$1 = $2$3", null, 0),
            // (ref left)
            // (left)
            (new Regex(@"\(ref ([a-zA-Z0-9]+)(\)|\(|,)"), "($1$2", null, 0),
            //  ref TElement
            //  TElement*
            (new Regex(@"( |\()ref ([a-zA-Z0-9]+) "), "$1$2* ", null, 0),
            // ref sizeBalancedTree.Root
            // &sizeBalancedTree->Root
            (new Regex(@"ref ([a-zA-Z0-9]+)\.([a-zA-Z0-9\*]+)"), "&$1->$2", null, 0),
            // ref GetElement(node).Right
            // &GetElement(node)->Right
            (new Regex(@"ref ([a-zA-Z0-9]+)\(([a-zA-Z0-9\*]+)\)\.([a-zA-Z0-9]+)"), "&$1($2)->$3", null, 0),
            // GetElement(node).Right
            // GetElement(node)->Right
            (new Regex(@"([a-zA-Z0-9]+)\(([a-zA-Z0-9\*]+)\)\.([a-zA-Z0-9]+)"), "$1($2)->$3", null, 0),
            // [Fact]\npublic: static void SizeBalancedTreeMultipleAttachAndDetachTest()
            // public: TEST_METHOD(SizeBalancedTreeMultipleAttachAndDetachTest)
            (new Regex(@"\[Fact\][\s\n]+(public: )?(static )?void ([a-zA-Z0-9]+)\(\)"), "public: TEST_METHOD($3)", null, 0),
            // class TreesTests
            // TEST_CLASS(TreesTests)
            (new Regex(@"class ([a-zA-Z0-9]+)Tests"), "TEST_CLASS($1)", null, 0),
            // Assert.Equal
            // Assert::AreEqual
            (new Regex(@"Assert\.Equal"), "Assert::AreEqual", null, 0),
            // $"Argument {argumentName} is null."
            // ((std::string)"Argument ").append(argumentName).append(" is null.").data()
            (new Regex(@"\$""(?<left>(\\""|[^""\r\n])*){(?<expression>[_a-zA-Z0-9]+)}(?<right>(\\""|[^""\r\n])*)"""),
            "((std::string)$\"${left}\").append(${expression}).append(\"${right}\").data()", null, 10),
            // $"
            // "
            (new Regex(@"\$"""), "\"", null, 0),
            // Console.WriteLine("...")
```

```
            // printf("...\n")
            (new Regex(@"Console\.WriteLine\(""([^""\r\n]+)""\)"), "printf(\"$1\\n\")", null, 0),
            // TElement Root;
            // TElement Root = 0;
            (new Regex(@"(\r?\n[\t ]+)(private|protected|public)?(:
            ↪  )?([a-zA-Z0-9:_]+(?<!return)) ([_a-zA-Z0-9]+);"), "$1$2$3$4 $5 = 0;", null, 0),
            // TreeElement _elements[N];
            // TreeElement _elements[N] = { {0} };
            (new Regex(@"(\r?\n[\t ]+)(private|protected|public)?(: )?([a-zA-Z0-9]+)
            ↪  ([_a-zA-Z0-9]+)\[([_a-zA-Z0-9]+)\];"), "$1$2$3$4 $5[$6] = { {0} };", null, 0),
            // auto path = new TElement[MaxPath];
            // TElement path[MaxPath] = { {0} };
            (new Regex(@"(\r?\n[\t ]+)[a-zA-Z0-9]+ ([a-zA-Z0-9]+) = new
            ↪  ([a-zA-Z0-9]+)\[([_a-zA-Z0-9]+)\];"), "$1$3 $2[$4] = { {0} };", null, 0),
            // Insert scope borders.
            // auto added = new StringBuilder();
            // /*~sb~*/std::string added;
            (new Regex(@"(auto|(System\.Text\.)?StringBuilder) (?<variable>[a-zA-Z0-9]+) = new
            ↪  (System\.Text\.)?StringBuilder\(\);"), "/*~${variable}~*/std::string
            ↪  ${variable};", null, 0),
            // static void Indent(StringBuilder sb, int level)
            // static void Indent(/*~sb~*/StringBuilder sb, int level)
            (new Regex(@"(?<start>, |\()(System\.Text\.)?StringBuilder
            ↪  (?<variable>[a-zA-Z0-9]+)(?<end>,|\))"), "${start}/*~${variable}~*/std::string&
            ↪  ${variable}${end}", null, 0),
            // Inside the scope of ~!added!~ replace:
            // sb.ToString()
            // sb.data()
            (new Regex(@"(?<scope>/\*~(?<variable>[a-zA-Z0-9]+)~\*/)(?<separator>.|\n)(?<before>
            ↪  ((?<!/\*~\k<variable>~\*/)(.|\n))*?)\k<variable>\.ToString\(\)"),
            ↪  "${scope}${separator}${before}${variable}.data()", null, 10),
            // sb.AppendLine(argument)
            // sb.append(argument).append('\n')
            (new Regex(@"(?<scope>/\*~(?<variable>[a-zA-Z0-9]+)~\*/)(?<separator>.|\n)(?<before>
            ↪  ((?<!/\*~\k<variable>~\*/)(.|\n))*?)\k<variable>\.AppendLine\((?<argument>[^\),\
            ↪  r\n]+)\)"),
            ↪  "${scope}${separator}${before}${variable}.append(${argument}).append('\\n')",
            ↪  null, 10),
            // sb.Append('\t', level);
            // sb.append(level, '\t');
            (new Regex(@"(?<scope>/\*~(?<variable>[a-zA-Z0-9]+)~\*/)(?<separator>.|\n)(?<before>
            ↪  ((?<!/\*~\k<variable>~\*/)(.|\n))*?)\k<variable>\.Append\('(?<character>[^'\r\n]
            ↪  +)', (?<count>[^\),\r\n]+)\)"),
            ↪  "${scope}${separator}${before}${variable}.append(${count}, '${character}')",
            ↪  null, 10),
            // sb.Append(argument)
            // sb.append(argument)
            (new Regex(@"(?<scope>/\*~(?<variable>[a-zA-Z0-9]+)~\*/)(?<separator>.|\n)(?<before>
            ↪  ((?<!/\*~\k<variable>~\*/)(.|\n))*?)\k<variable>\.Append\((?<argument>[^\),\r\n]
            ↪  +)\)"), "${scope}${separator}${before}${variable}.append(${argument})", null,
            ↪  10),
            // Remove scope borders.
            // /*~sb~*/
            //
            (new Regex(@"/\*~(?<pointer>[a-zA-Z0-9]+)~\*/"), "", null, 0),
            // Insert scope borders.
            // auto added = new HashSet<TElement>();
            // ~!added!~std::unordered_set<TElement> added;
            (new Regex(@"auto (?<variable>[a-zA-Z0-9]+) = new
            ↪  HashSet<(?<element>[a-zA-Z0-9]+)>\(\);"),
            ↪  "~!${variable}!~std::unordered_set<${element}> ${variable};", null, 0),
            // Inside the scope of ~!added!~ replace:
            // added.Add(node)
            // added.insert(node)
            (new Regex(@"(?<scope>~!(?<variable>[a-zA-Z0-9]+)!~)(?<separator>.|\n)(?<before>((?<
            ↪  !~!\k<variable>!~)(.|\n))*?)\k<variable>\.Add\((?<argument>[a-zA-Z0-9]+)\)"),
            ↪  "${scope}${separator}${before}${variable}.insert(${argument})", null, 10),
            // Inside the scope of ~!added!~ replace:
            // added.Remove(node)
            // added.erase(node)
            (new Regex(@"(?<scope>~!(?<variable>[a-zA-Z0-9]+)!~)(?<separator>.|\n)(?<before>((?<
            ↪  !~!\k<variable>!~)(.|\n))*?)\k<variable>\.Remove\((?<argument>[a-zA-Z0-9]+)\)"),
            ↪  "${scope}${separator}${before}${variable}.erase(${argument})", null, 10),
            // if (added.insert(node)) {
            // if (!added.contains(node)) { added.insert(node);
```

```
269             (new Regex(@"if \((?<variable>[a-zA-Z0-9]+)\.insert\((?<argument>[a-zA-Z0-9]+)\)\)(?
                ↪  <separator>[\t ]*[\r\n]+)(?<indent>[\t ]*){"), "if
                ↪  (!${variable}.contains(${argument}))${separator}${indent}{" +
                ↪  Environment.NewLine + "${indent}    ${variable}.insert(${argument});", null, 0),
270             // Remove scope borders.
271             // ~!added!~
272             //
273             (new Regex(@"~!(?<pointer>[a-zA-Z0-9]+)!~"), "", null, 5),
274             // Insert scope borders.
275             // auto random = new System.Random(0);
276             // std::srand(0);
277             (new Regex(@"[a-zA-Z0-9\.]+ ([a-zA-Z0-9]+) = new
                ↪  (System\.)?Random\(([a-zA-Z0-9]+)\);"), "~!$1!~std::srand($3);", null, 0),
278             // Inside the scope of ~!random!~ replace:
279             // random.Next(1, N)
280             // (std::rand() % N) + 1
281             (new Regex(@"(?<scope>~!(?<variable>[a-zA-Z0-9]+)!~)(?<separator>.|\n)(?<before>((?<
                ↪  !~!\k<variable>!~)(.|\n))*?)\k<variable>\.Next\((?<from>[a-zA-Z0-9]+),
                ↪  (?<to>[a-zA-Z0-9]+)\)"), "${scope}${separator}${before}(std::rand() % ${to}) +
                ↪  ${from}", null, 10),
282             // Remove scope borders.
283             // ~!random!~
284             //
285             (new Regex(@"~!(?<pointer>[a-zA-Z0-9]+)!~"), "", null, 5),
286             // Insert method body scope starts.
287             // void PrintNodes(TElement node, StringBuilder sb, int level) {
288             // void PrintNodes(TElement node, StringBuilder sb, int level) {/*method-start*/
289             (new Regex(@"(?<start>\r?\n[\t ]+)(?<prefix>((private|protected|public): )?(virtual
                    )?[a-zA-Z0-9:_]+
                ↪  )?(?<method>[a-zA-Z][a-zA-Z0-9]*)\((?<arguments>[^\)]*)\)(?<override>(
                ↪  override)?)(?<separator>[ \t\r\n]*)\{(?<end>[^~])"), "${start}${prefix}${method}
                ↪  (${arguments})${override}${separator}{/*method-start*/${end}", null,
                ↪  0),
290             // Insert method body scope ends.
291             // {/*method-start*/...}
292             // {/*method-start*/.../*method-end*/}
293             (new Regex(@"\{/\*method-start\*/(?<body>((?<bracket>\{)|(?<-bracket>\})|[^\{\}]*)+)
                ↪  \}"), "{/*method-start*/${body}/*method-end*/}", null,
                ↪  0),
294             // Inside method bodies replace:
295             // GetFirst(
296             // this->GetFirst(
297             //(new Regex(@"(?<separator>(\(|, |([\W]) |return ))(?<!(->|\*
                ↪  ))(?<method>(?!sizeof)[a-zA-Z0-9]+)\((?!\) \{)"),
                ↪  "${separator}this->${method}(", null, 1),
298             (new Regex(@"(?<scope>/\*method-start\*/)(?<before>((?<!/\*method-end\*/)(.|\n))*?)(
                ↪  ?<separator>[\W](?<!(::|\.|->)))(?<method>(?!sizeof)[a-zA-Z0-9]+)\((?!\)
                ↪  \{)(?<after>(.|\n)*?)(?<scopeEnd>/\*method-end\*/)"),
                ↪  "${scope}${before}${separator}this->${method}(${after}${scopeEnd}", null, 100),
299             // Remove scope borders.
300             // /*method-start*/
301             //
302             (new Regex(@"/\*method-(start|end)\*/"), "", null, 0),
303             // throw new ArgumentNullException(argumentName, message);
304             // throw std::invalid_argument(((std::string)"Argument
                ↪  ").append(argumentName).append(" is null: ").append(message).append("."));
305             (new Regex(@"throw new
                    ArgumentNullException\((?<argument>[a-zA-Z]*[Aa]rgument[a-zA-Z]*),
                ↪  (?<message>[a-zA-Z]*[Mm]essage[a-zA-Z]*)\);"), "throw
                ↪  std::invalid_argument(((std::string)\"Argument \").append(${argument}).append(\"
                ↪  is null: \").append(${message}).append(\".\"));", null, 0),
306             // throw new ArgumentException(message, argumentName);
307             // throw std::invalid_argument(((std::string)"Invalid
                ↪  ").append(argumentName).append(" argument: ").append(message).append("."));
308             (new Regex(@"throw new ArgumentException\((?<message>[a-zA-Z]*[Mm]essage[a-zA-Z]*),
                    (?<argument>[a-zA-Z]*[Aa]rgument[a-zA-Z]*)\);"), "throw
                ↪  std::invalid_argument(((std::string)\"Invalid \").append(${argument}).append(\"
                ↪  argument: \").append(${message}).append(\".\"));", null, 0),
309             // throw new NotSupportedException();
310             // throw std::logic_error("Not supported exception.");
311             (new Regex(@"throw new NotSupportedException\(\);"), "throw std::logic_error(\"Not
                ↪  supported exception.\");", null, 0),
312             // throw new NotImplementedException();
313             // throw std::logic_error("Not implemented exception.");
314             (new Regex(@"throw new NotImplementedException\(\);"), "throw std::logic_error(\"Not
                ↪  implemented exception.\");", null, 0),
```

```
        }.Cast<ISubstitutionRule>().ToList();

        public static readonly IList<ISubstitutionRule> LastStage = new List<SubstitutionRule>
        {
            // ICounter<int, int> c1;
            // ICounter<int, int>* c1;
            (new Regex(@"(?<abstractType>I[A-Z][a-zA-Z0-9]+(<[^>\r\n]+>)?)
            ↪  (?<variable>[_a-zA-Z0-9]+);"), "${abstractType}* ${variable};", null, 0),
            // (expression)
            // expression
            (new Regex(@"(\(| )\(([a-zA-Z0-9_\*:]+)\)(,| |;|\))"), "$1$2$3", null, 0),
            // (method(expression))
            // method(expression)
            (new Regex(@"(?<firstSeparator>(\(|
            ↪  ))\((?<method>[a-zA-Z0-9_\->\*:]+)\((?<expression>((?<parenthesis>\()|(?<-parent
            ↪  hesis>\))|[a-zA-Z0-9_\->\*:]*)+)(?(parenthesis)(?!))\)\)(?<lastSeparator>(,|
            ↪  |;|\)))"), "${firstSeparator}${method}(${expression})${lastSeparator}", null, 0),
            // return ref _elements[node];
            // return &_elements[node];
            (new Regex(@"return ref ([_a-zA-Z0-9]+)\[([_a-zA-Z0-9\*]+)\];"), "return &$1[$2];",
            ↪  null, 0),
            // null
            // NULL
            (new Regex(@"(?<before>\r?\n[^""\r\n]*(""(\\""|[^""\r\n])*""[^""\r\n]*)*)(?<=\W)null
            ↪  (?<after>\W)"), "${before}NULL${after}", null,
            ↪  10),
            // default
            // 0
            (new Regex(@"(?<before>\r?\n[^""\r\n]*(""(\\""|[^""\r\n])*""[^""\r\n]*)*)(?<=\W)defa
            ↪  ult(?<after>\W)"), "${before}0${after}", null,
            ↪  10),
            // #region Always
            //
            (new Regex(@"(^|\r?\n)[ \t]*\#(region|endregion)[^\r\n]*(\r?\n|$)"), "", null, 0),
            // //#define ENABLE_TREE_AUTO_DEBUG_AND_VALIDATION
            //
            (new Regex(@"\/\/[ \t]*\#define[ \t]+[_a-zA-Z0-9]+[ \t]*"), "", null, 0),
            // #if USEARRAYPOOL\r\n#endif
            //
            (new Regex(@"#if [a-zA-Z0-9]+\s+#endif"), "", null, 0),
            // [Fact]
            //
            (new Regex(@"(?<firstNewLine>\r?\n|\A)(?<indent>[\t
            ↪  ]+)\[[a-zA-Z0-9]+(\((?<expression>((?<parenthesis>\()|(?<-parenthesis>\))|[^()\r
            ↪  \n]*)+)(?(parenthesis)(?!))\))?\][ \t]*(\r?\n\k<indent>)?"),
            ↪  "${firstNewLine}${indent}", null, 5),
            // \n ... namespace
            // namespace
            (new Regex(@"(\S[\r\n]{1,2})?[\r\n]+namespace"), "$1namespace", null, 0),
            // \n ... class
            // class
            (new Regex(@"(\S[\r\n]{1,2})?[\r\n]+class"), "$1class", null, 0),
        }.Cast<ISubstitutionRule>().ToList();

        public CSharpToCppTransformer(IList<ISubstitutionRule> extraRules) :
        ↪  base(FirstStage.Concat(extraRules).Concat(LastStage).ToList()) { }

        public CSharpToCppTransformer() : base(FirstStage.Concat(LastStage).ToList()) { }
    }
}
```

1.2 ./Platform.RegularExpressions.Transformer.CSharpToCpp.Tests/CSharpToCppTransformerTests.cs

```
using Xunit;

namespace Platform.RegularExpressions.Transformer.CSharpToCpp.Tests
{
    public class CSharpToCppTransformerTests
    {
        [Fact]
        public void EmptyLineTest()
        {
            // This test can help to test basic problems with regular expressions like incorrect
            ↪  syntax
            var transformer = new CSharpToCppTransformer();
            var actualResult = transformer.Transform("", new Context(null));
            Assert.Equal("", actualResult);
        }

```

```
        [Fact]
        public void HelloWorldTest()
        {
            const string helloWorldCode = @"using System;
class Program
{
    public static void Main(string[] args)
    {
        Console.WriteLine(""Hello, world!"");
    }
}";
            const string expectedResult = @"class Program
{
    public: static void Main(const char* args[])
    {
        printf(""Hello, world!\n"");
    }
};";
            var transformer = new CSharpToCppTransformer();
            var actualResult = transformer.Transform(helloWorldCode, new Context(null));
            Assert.Equal(expectedResult, actualResult);
        }
    }
}
```

Index