./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),
            // public abstract class
            // class
            (new Regex(@"(public abstract|static) class"), "class", null, 0),
            // class GenericCollectionMethodsBase {
            // class GenericCollectionMethodsBase { public:
            (new Regex(@"class ([a-zA-Z0-9]+)(\s+){"), "class $1$2{" + Environment.NewLine + "    public:", null, 0),
            // class GenericCollectionMethodsBase<TElement> {
            // template <typename TElement> class GenericCollectionMethodsBase { public:
            (new Regex(@"class ([a-zA-Z0-9]+)<([a-zA-Z0-9]+)>([^{]+){"), "template <typename $2> class $1$3{" + Environment.NewLine + "    public:", 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]+)>\(([^\)]+)\)"), "template <typename $3> static $1 $2($4)", null, 0),
            // (this
            // (
            (new Regex(@"\(this "), "(", null, 0),
            // Func<TElement> treeCount
            // std::function<TElement()> treeCount
            (new Regex(@"Func<([a-zA-Z0-9]+)> ([a-zA-Z0-9]+)"), "std::function<$1()> $2", null, 0),
            // Action<TElement> free
            // std::function<void(TElement)> free
            (new Regex(@"Action<([a-zA-Z0-9]+)> ([a-zA-Z0-9]+)"), "std::function<void($1)> $2", null, 0),
            // private const int MaxPath = 92;
            // static const int MaxPath = 92;
            (new Regex(@"private const ([a-zA-Z0-9]+) ([_a-zA-Z0-9]+) = ([a-zA-Z0-9]+);"), "static const $1 $2 = $3;", null, 0),
            // protected virtual
            // virtual
            (new Regex(@"protected virtual"), "virtual", null, 0),
            // protected abstract TElement GetFirst();
            // virtual TElement GetFirst() = 0;
            (new Regex(@"protected abstract ([^;]+);"), "virtual $1 = 0;", null, 0),
            // public virtual
            // virtual
            (new Regex(@"public virtual"), "virtual", null, 0),
            // protected readonly
            //
            (new Regex(@"protected readonly "), "", null, 0),
            // protected readonly TreeElement[] _elements;
            // TreeElement _elements[N];
```

```
            (new Regex(@"(protected|private) readonly ([a-zA-Z<>0-9]+)([\[\]]+)
            ↪  ([_a-zA-Z0-9]+);"), "$2 $4[N];", null, 0),
            // protected readonly TElement Zero;
            // TElement Zero;
            (new Regex(@"(protected|private) readonly ([a-zA-Z<>0-9]+) ([_a-zA-Z0-9]+);"), "$2
            ↪  $3;", null, 0),
            // private
            //
            (new Regex(@"(\W)(private|protected|public|internal) "), "$1", null, 0),
            // SizeBalancedTree(int capacity) => a = b;
            // SizeBalancedTree(int capacity) { a = b; }
            (new Regex(@"(^\s+)(override )?(void )?([a-zA-Z0-9]+)\(([^\(]+)\)\s+=>\s+([^;]+);"),
            ↪  "$1$2$3$4($5) { $6; }", null, 0),
            // () => Integer<TElement>.Zero,
            // () { return Integer<TElement>.Zero; },
            (new Regex(@"\(\)\s+=>\s+([^\r\n,;]+?),"), "() { return $1; },", null, 0),
            // => Integer<TElement>.Zero;
            // { return Integer<TElement>.Zero; }
            (new Regex(@"\)\s+=>\s+([^\r\n;]+?);"), ") { return $1; }", null, 0),
            // () { return avlTree.Count; }
            // [&]()-> auto { return avlTree.Count; }
            (new Regex(@", \(\) { return ([^;]+); }"), ", [&]()-> auto { return $1; }", null, 0),
            // Count => GetSizeOrZero(Root);
            // GetCount() { return GetSizeOrZero(Root); }
            (new Regex(@"([A-Z][a-z]+)\s+=>\s+([^;]+);"), "Get$1() { return $2; }", null, 0),
            // var
            // auto
            (new Regex(@"(\W)var(\W)"), "$1auto$2", null, 0),
            // unchecked
            //
            (new Regex(@"[\r\n]{2}\s*?unchecked\s*?$"), "", null, 0),
            // $"
            // "
            (new Regex(@"\$"""), "\"", null, 0),
            // Console.WriteLine("...")
            // printf("...\n")
            (new Regex(@"Console\.WriteLine\(""([^""]+)""\)"), "printf(\"$1\\n\")", null, 0),
            // throw new InvalidOperationException
            // throw std::exception
            (new Regex(@"throw new (InvalidOperationException|Exception)"), "throw
            ↪  std::exception", null, 0),
            // override void PrintNode(TElement node, StringBuilder sb, int level)
            // void PrintNode(TElement node, StringBuilder sb, int level) override
            (new Regex(@"override ([a-zA-Z0-9 \*\+]+)(\([^\)]+?\))"), "$1$2 override", null, 0),
            // string
            // char*
            (new Regex(@"(\W)string(\W)"), "$1char*$2", null, 0),
            // sbyte
            // std::int8_t
            (new Regex(@"(\W)sbyte(\W)"), "$1std::int8_t$2", null, 0),
            // uint
            // std::uint32_t
            (new Regex(@"(\W)uint(\W)"), "$1std::uint32_t$2", null, 0),
            // char*[] args
            // char* args[]
            (new Regex(@"([_a-zA-Z0-9:\*]?)\[\] ([a-zA-Z0-9]+)"), "$1 $2[]", null, 0),
            // using Platform.Numbers;
            //
            (new Regex(@"([\r\n]{2}|^)\s*?using [\.a-zA-Z0-9]+;\s*?$"), "", null, 0),
            // struct TreeElement { }
            // 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),
            // 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),
```

```
132             // Inside the scope of ~!root!~ replace:
133             // root
134             // *root
135             (new Regex(@"(?<definition>~!(?<pointer>[a-zA-Z0-9]+)!~ref [a-zA-Z0-9]+
                    (?<pointer>[a-zA-Z0-9]+)(?=\)|, |
                =))(?<before>((?<!~!\k<pointer>!~)(.|\n))*?)(?<prefix>(\W
                |\())\k<pointer>(?<suffix>( |\)|;|,))"),
                "${definition}${before}${prefix}*${pointer}${suffix}", null, 70),
136             // Remove scope borders.
137             // ~!root!~
138             //
139             (new Regex(@"~!(?<pointer>[a-zA-Z0-9]+)!~"), "", null, 5),
140             // ref auto root = ref
141             // ref auto root =
142             (new Regex(@"ref ([a-zA-Z0-9]+) ([a-zA-Z0-9]+) = ref(\W)"), "$1* $2 =$3", null, 0),
143             // *root = ref left;
144             // root = left;
145             (new Regex(@"\*([a-zA-Z0-9]+) = ref ([a-zA-Z0-9]+)(\W)"), "$1 = $2$3", null, 0),
146             // (ref left)
147             // (left)
148             (new Regex(@"\(ref ([a-zA-Z0-9]+)(\)|\(|,)"), "($1$2", null, 0),
149             //  ref TElement
150             //  TElement*
151             (new Regex(@"( |\()ref ([a-zA-Z0-9]+) "), "$1$2* ", null, 0),
152             // ref sizeBalancedTree2.Root
153             // &sizeBalancedTree2->Root
154             (new Regex(@"ref ([a-zA-Z0-9]+)\.([a-zA-Z0-9\*]+)"), "&$1->$2", null, 0),
155             // ref GetElement(node).Right
156             // &GetElement(node)->Right
157             (new Regex(@"ref ([a-zA-Z0-9]+)\(([a-zA-Z0-9\*]+)\)\.([a-zA-Z0-9]+)"),
                "&$1($2)->$3", null, 0),
158             // GetElement(node).Right
159             // GetElement(node)->Right
160             (new Regex(@"([a-zA-Z0-9]+)\(([a-zA-Z0-9\*]+)\)\.([a-zA-Z0-9]+)"), "$1($2)->$3",
                null, 0),
161             // [Fact]\npublic static void SizeBalancedTreeMultipleAttachAndDetachTest()
162             // TEST_METHOD(SizeBalancedTreeMultipleAttachAndDetachTest)
163             (new Regex(@"\[Fact\][\s\n]+(static )?void ([a-zA-Z0-9]+)\(\)"), "TEST_METHOD($2)",
                null, 0),
164             // class TreesTests
165             // TEST_CLASS(TreesTests)
166             (new Regex(@"class ([a-zA-Z0-9]+)Tests"), "TEST_CLASS($1)", null, 0),
167             // Assert.Equal
168             // Assert::AreEqual
169             (new Regex(@"Assert\.Equal"), "Assert::AreEqual", null, 0),
170             // TElement Root;
171             // TElement Root = 0;
172             (new Regex(@"(\r?\n[\t ]+)([a-zA-Z0-9:_]+(?<!return)) ([_a-zA-Z0-9]+);"), "$1$2 $3 =
                0;", null, 0),
173             // TreeElement _elements[N];
174             // TreeElement _elements[N] = { {0} };
175             (new Regex(@"(\r?\n[\t ]+)([a-zA-Z0-9]+) ([_a-zA-Z0-9]+)\[([_a-zA-Z0-9]+)\];"),
                "$1$2 $3[$4] = { {0} };", null, 0),
176             // auto path = new TElement[MaxPath];
177             // TElement path[MaxPath] = { {0} };
178             (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),
179             // Insert scope borders.
180             // auto added = new HashSet<TElement>();
181             // ~!added!~std::unordered_set<TElement> added;
182             (new Regex(@"auto (?<variable>[a-zA-Z0-9]+) = new
                HashSet<(?<element>[a-zA-Z0-9]+)>\(\);"),
                "~!${variable}!~std::unordered_set<${element}> ${variable};", null, 0),
183             // Inside the scope of ~!added!~ replace:
184             // added.Add(node)
185             // added.insert(node)
186             (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),
187             // Inside the scope of ~!added!~ replace:
188             // added.Remove(node)
189             // added.erase(node)
190             (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),
191             // if (added.insert(node)) {
```

```
            // if (!added.contains(node)) { added.insert(node);
            (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),
            // Remove scope borders.
            // ~!added!~
            //
            (new Regex(@"~!(?<pointer>[a-zA-Z0-9]+)!~"), "", null, 5),
            // Insert scope borders.
            // auto random = new System.Random(0);
            // std::srand(0);
            (new Regex(@"[a-zA-Z0-9\.]+ ([a-zA-Z0-9]+) = new
                (System\.)?Random\(([a-zA-Z0-9]+)\);"), "~!$1!~std::srand($3);", null, 0),
            // Inside the scope of ~!random!~ replace:
            // random.Next(1, N)
            // (std::rand() % N) + 1
            (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),
            // Remove scope borders.
            // ~!random!~
            //
            (new Regex(@"~!(?<pointer>[a-zA-Z0-9]+)!~"), "", null, 5),
        }.Cast<ISubstitutionRule>().ToList();

        public static readonly IList<ISubstitutionRule> LastStage = new List<SubstitutionRule>
        {
            // (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[$2];",
                null, 0),
            // default
            // 0
            (new Regex(@"(\W)default(\W)"), "${1}0$2", 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>\()|(?<-parenthesis>\))|[^()]*
                )+)(?(parenthesis)(?!))\))?\]\s*(\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()) { }
    }
}
```

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

```
using Xunit;

namespace Platform.RegularExpressions.Transformer.CSharpToCpp.Tests
{
```

```
    public class CSharpToCppTransformerTests
    {
        [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(char* args[])
    {
        printf(""Hello, world!\n"");
    }
};";
            var transformer = new CSharpToCppTransformer();
            var actualResult = transformer.Transform(helloWorldCode, new Context(null));
            Assert.Equal(expectedResult, actualResult);
        }
    }
}
```

Index