

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text.RegularExpressions;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.RegularExpressions.Transformer.CSharpToCpp
9  {
10     public class CSharpToCppTransformer : Transformer
11     {
12         public static readonly IList<ISubstitutionRule> FirstStage = new List<SubstitutionRule>
13         {
14             // // ...
15             //
16             (new Regex(@"(\r?\n)?[ \t]+//+.+"), "", null, 0),
17             // #pragma warning disable CS1591 // Missing XML comment for publicly visible type
18             // or member
19             //
20             (new Regex(@"^-s*?#pragma\[sa-zA-Z0-9]+\$"), "", null, 0),
21             // {\n\n\n
22             // {
23             (new Regex(@"{\s+[\r\n]+") , "{" + Environment.NewLine, null, 0),
24             // Platform.Collections.Methods.Lists
25             // Platform::Collections::Methods::Lists
26             (new Regex(@"(namespace[^\r\n]+?)\.([^\r\n]+?)"), "$1::$2", null, 20),
27             // out TProduct
28             // TProduct
29             (new Regex(@"(<?before>(<|,|))<in|out>")
30             → (<?typeParameter>[a-zA-Z0-9]+)<?after>(>|,|))"),
31             → "{$before}$<typeParameter>$<after>", null, 10),
32             // public ...
33             // public: ...
34             (new Regex(@"(<?newLineAndIndent>\r?\n?[
35             → \t]*)<?before>[^\{\\(\r\n)*]<?access>private|protected|public)[
36             → \t]+<?![^\{\\(\r\n)*<(interface|class|struct)<[^\{\\(\r\n)*[^\{\\(\r\n)*)>"),
37             → "{$newLineAndIndent}$<access>: {$before}", null, 0),
38             // public: static bool CollectExceptions { get; set; }
39             // public: static bool CollectExceptions;
40             (new Regex(@"(<?before>(private|protected|public): (static )?<?[\r\n]+
41             → )(<?name>[a-zA-Z0-9]+) {[^;]}*(<?=<W>get;[^;]}*(<?=<W>set;[^;]}*") ,
42             → "{$before}$<name>;", null, 0),
43             // public abstract class
44             // class
45             (new Regex(@"((public|protected|private|internal|abstract|static)
46             → )*<?category>interface|class|struct") , "{$category}", null, 0),
47             // class GenericCollectionMethodsBase<TElement> {
48             // template <typename TElement> class GenericCollectionMethodsBase {
49             (new Regex(@"class ([a-zA-Z0-9]+)<([a-zA-Z0-9]+)><([^\{]+)>{") , "template <typename $2>
50             → class $1$3{", null, 0),
51             // static void
52             → TestMultipleCreationsAndDeletions<TElement>(SizedBinaryTreeMethodsBase<TElement>
53             → tree, TElement* root)
54             // template<typename T> static void
55             → TestMultipleCreationsAndDeletions<TElement>(SizedBinaryTreeMethodsBase<TElement>
56             → tree, TElement* root)
57             (new Regex(@"static ([a-zA-Z0-9]+) ([a-zA-Z0-9]+)<([a-zA-Z0-9]+)><((([^\r\n]+)\r\n)+)\>") ,
58             → "template <typename $3> static $1 $2($4)", null, 0),
59             // interface IFactory<out TProduct> {
60             // template <typename TProduct> class IFactory { public:
61             (new Regex(@"interface (<?interface>[a-zA-Z0-9]+)<(<?<typeParameters>[a-zA-Z0-9-
62             → ,]+><?whitespace>[^\{]+)>{") , "template <typename...> class {$interface};
63             → template <typename $<typeParameters>> class
64             → {$interface}<$<typeParameters>>{$whitespace}{ + Environment.NewLine + "
65             → public:", null, 0),
66             // template <typename TObject, TProperty, TValue>
67             // template <typename TObject, typename TProperty, TValue>
68             (new Regex(@"(<?before>template <((,| )?typename [a-zA-Z0-9]+)+,
69             → )(<?typeParameter>[a-zA-Z0-9]+)<?after>(<|>))") , "{$before}<typename
70             → $<typeParameter>$<after>", null, 10),
71             // Insert markers
72             // private: static void BuildExceptionString(this StringBuilder sb, Exception
73             → exception, int level)
74             // /*~extensionMethod~BuildExceptionString~*/private: static void
75             → BuildExceptionString(this StringBuilder sb, Exception exception, int level)

```

```

53 (new Regex(@"private: static [^\r\n]+ (?<name>[a-zA-Z0-9]+)\(this [^\]\r\n]+\\)"),
54     ↳ "/*~extensionMethod~${name}~*/$0", null, 0),
55 // Move all markers to the beginning of the file.
56 (new Regex(@"\A(?<before>[^\r\n]+\r?\n(.|\n)+)(?<marker>/\*~extensionMethod~(?<name>[a-zA-Z0-9]+)~\*/)"), "${marker}${before}", null,
57     ↳ 10),
58 // /*~extensionMethod~BuildExceptionString~*/...sb.BuildExceptionString(exception.InnerException, level +
59     ↳ 1);
60 // /*~extensionMethod~BuildExceptionString~*/...BuildExceptionString(sb,
61     ↳ exception.InnerException, level + 1);
62 (new Regex(@"(?<before>/\*~extensionMethod~(?<name>[a-zA-Z0-9]+)~\*/(.|\n)+\W)(?<variable>[_a-zA-Z0-9]+\)\. \k<name>\(", "${before}${name}(${variable}", null,
63     ↳ 50),
64 // Remove markers
65 // /*~extensionMethod~BuildExceptionString~*/
66 //
67 (new Regex(@"/*~extensionMethod~[a-zA-Z0-9]+~\*/"), "", null, 0),
68 // (this
69 // (
70 (new Regex(@"\((this ", "(", null, 0),
71 // public: static readonly EnsureAlwaysExtensionRoot Always = new
72     ↳ EnsureAlwaysExtensionRoot();
73 // public: inline static EnsureAlwaysExtensionRoot Always;
74 (new Regex(@"(?<access>(private|protected|public): )?static readonly
75     ↳ (?<type>[a-zA-Z0-9]+) (?<name>[_a-zA-Z0-9_]+) = new \k<type>\(\);"),
76     ↳ "${access}inline static ${type} ${name};", null, 0),
77 // public: static readonly string ExceptionContentsSeparator = "---";
78 // public: inline static const char* ExceptionContentsSeparator = "---";
79 (new Regex(@"(?<access>(private|protected|public): )?static readonly string
80     ↳ (?<name>[a-zA-Z0-9_]+) = ""(?:<string>(\\"| [^\r\n]))+"";"), "${access}inline
81     ↳ static const char* ${name} = \"${string}\";", null, 0),
82 // private: const int MaxPath = 92;
83 // private: static const int MaxPath = 92;
84 (new Regex(@"(?<access>(private|protected|public): )?(const|static readonly)
85     ↳ (?<type>[a-zA-Z0-9]+) (?<name>[_a-zA-Z0-9_]+) = (?<value>[^\r\n]+);"),
86     ↳ "${access}static const ${type} ${name} = ${value};", null, 0),
87 // ArgumentNotNull(EnsureAlwaysExtensionRoot root, TArgument argument) where
88     ↳ TArgument : class
89 // ArgumentNotNull(EnsureAlwaysExtensionRoot root, TArgument* argument)
90 (new Regex(@"(?<before> [a-zA-Z]+)(([a-zA-Z *,,]+, |))(?<type>[a-zA-Z]+)(?<after>(|
91     ↳ [a-zA-Z *,,]+)))[ \r\n]+where \k<type> : class)", "${before}${type}*${after}",
92     ↳ null, 0),
93 // protected: abstract TElement GetFirst();
94 // protected: virtual TElement GetFirst() = 0;
95 (new Regex(@"(?<access>(private|protected|public): )?abstract
96     ↳ (?<method>[^\r\n]+);"), "${access}virtual ${method} = 0;", null, 0),
97 // TElement GetFirst();
98 // virtual TElement GetFirst() = 0;
99 (new Regex(@"([^\r\n]+[ ]+)((?!return)[a-zA-Z0-9]+ [a-zA-Z0-9]+\([^\]\r\n]*\))(\;|
100     ↳ ]*\[^\r\n]+\)"), "$1virtual $2 = 0$3", null, 1),
101 // protected: readonly TreeElement[] _elements;
102 // protected: TreeElement _elements[N];
103 (new Regex(@"(?<access>(private|protected|public): )?readonly
104     ↳ (?<type>[a-zA-Z<0-9]+)(\[\\]+\)(?<name>[_a-zA-Z0-9_]+);"), "${access}${type}
105     ↳ ${name}[N];", null, 0),
106 // protected: readonly TElement Zero;
107 // protected: TElement Zero;
108 (new Regex(@"(?<access>(private|protected|public): )?readonly
109     ↳ (?<type>[a-zA-Z<0-9]+) (?<name>[_a-zA-Z0-9_]+);"), "${access}${type} ${name};",
110     ↳ null, 0),
111 // internal
112 //
113 (new Regex(@"(\W)internal\s+"), "$1", null, 0),
114 // static void NotImplementedException(ThrowExtensionRoot root) => throw new
115     ↳ NotImplementedException();
116 // static void NotImplementedException(ThrowExtensionRoot root) { return throw new
117     ↳ NotImplementedException(); }
118 (new Regex(@"(^\s+)(private|protected|public)?(: )?(template \<[^\r\n]+\> )?(static
119     ↳ )?(override )?([a-zA-Z0-9]+
120     ↳ )([a-zA-Z0-9]+\)\((( [^\]\r\n]*)\)\s+=>\s+throw([^\r\n]+);"),
121     ↳ "$1$2$3$4$5$6$7$8($9) { throw$10; }", null, 0),
122 // SizeBalancedTree(int capacity) => a = b;
123 // 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 { };

```

```

161 (new Regex(@"(struct|class) ([a-zA-Z0-9]+)(\s+){([\sa-zA-Z0-9;:_]+?)}([\^;])", "$1
    ↳ $2$3{$4};$5", null, 0),
162 // class Program { }
163 // class Program { };
164 (new Regex(@"(struct|class) ([a-zA-Z0-9]+)[^\r\n]*)([\r\n]+(?<indentLevel>[\t
    ↳ ]*)?)\{([\S\s]+?[\r\n]+\k<indentLevel>\}\}([\^;]|$)", "$1 $2$3{$4};$5", null, 0),
165 // class SizedBinaryTreeMethodsBase : GenericCollectionMethodsBase
166 // class SizedBinaryTreeMethodsBase : public GenericCollectionMethodsBase
167 (new Regex(@"class ([a-zA-Z0-9]+) : ([a-zA-Z0-9]+)", "class $1 : public $2", null,
    ↳ 0),
168 // class IProperty : ISetter<TValue, TObject>, IProvider<TValue, TObject>
169 // class IProperty : public ISetter<TValue, TObject>, IProvider<TValue, TObject>
170 (new Regex(@"(?<before>class [a-zA-Z0-9]+ : ((public [a-zA-Z0-9]+(<[a-zA-Z0-9
    ↳ ,]+>)?, )+)?(?<inheritedType>(?!public) [a-zA-Z0-9]+(<[a-zA-Z0-9
    ↳ ,]+>)?(?<after>(, [a-zA-Z0-9]+(?!>)|[\r\n]+)))", "${before}public
    ↳ ${inheritedType}${after}", null, 10),
171 // Insert scope borders.
172 // ref TELEMENT root
173 // ~!root!~ref TELEMENT root
174 (new Regex(@"(?<definition>(?!<= |\\() (ref [a-zA-Z0-9]+|[a-zA-Z0-9]+(?<ref))
    ↳ (?<variable>[a-zA-Z0-9]+)(?=\\|, | =))", "~!${variable}!~${definition}", null,
    ↳ 0),
175 // Inside the scope of ~!root!~ replace:
176 // root
177 // *root
178 (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),
179 // Remove scope borders.
180 // ~!root!~
181 //
182 (new Regex(@"~!(?<pointer>[a-zA-Z0-9]+)!~"), "", null, 5),
183 // ref auto root = ref
184 // ref auto root =
185 (new Regex(@"ref ([a-zA-Z0-9]+) ([a-zA-Z0-9]+) = ref(\\W)", "$1* $2 =$3", null, 0),
186 // *root = ref left;
187 // root = left;
188 (new Regex(@"*([a-zA-Z0-9]+) = ref ([a-zA-Z0-9]+)(\\W)", "$1 = $2$3", null, 0),
189 // (ref left)
190 // (left)
191 (new Regex(@"\\(ref ([a-zA-Z0-9]+)(\\|\\(|,))", "($1$2", null, 0),
192 // ref TELEMENT
193 // TELEMENT*
194 (new Regex(@"( |\\()ref ([a-zA-Z0-9]+) ", "$1$2* ", null, 0),
195 // ref sizeBalancedTree.Root
196 // &sizeBalancedTree->Root
197 (new Regex(@"ref ([a-zA-Z0-9]+)\\.([a-zA-Z0-9\\*]+)", "&$1->$2", null, 0),
198 // ref GetElement(node).Right
199 // &GetElement(node)->Right
200 (new Regex(@"ref ([a-zA-Z0-9]+)\\((([a-zA-Z0-9\\*]+)\\)\\.([a-zA-Z0-9]+)",
    ↳ "&$1($2)->$3", null, 0),
201 // GetElement(node).Right
202 // GetElement(node)->Right
203 (new Regex(@"([a-zA-Z0-9]+)\\((([a-zA-Z0-9\\*]+)\\)\\.([a-zA-Z0-9]+)", "$1($2)->$3",
    ↳ null, 0),
204 // [Fact] npublic: static void SizeBalancedTreeMultipleAttachAndDetachTest()
205 // public: TEST_METHOD(SizeBalancedTreeMultipleAttachAndDetachTest)
206 (new Regex(@"\\[Fact\\][\\s\\n]+(public: )?(static )?void ([a-zA-Z0-9]+)\\(\\)", "public:
    ↳ TEST_METHOD($3)", null, 0),
207 // class TreesTests
208 // TEST_CLASS(TreesTests)
209 (new Regex(@"class ([a-zA-Z0-9]+)Tests", "TEST_CLASS($1)", null, 0),
210 // Assert.Equal
211 // Assert::AreEqual
212 (new Regex(@"Assert\\.Equal", "Assert::AreEqual", null, 0),
213 // $Argument {argumentName} is null."
214 // ((std::string)"Argument ").append(argumentName).append(" is null.").data()
215 (new Regex(@"\\$""(?<left>(\\""|\\^""\\r\\n)*){(?<expression>[_a-zA-Z0-9]+)}{(?<right>(\\
    ↳ \\""|\\^""\\r\\n)*)""",
    ↳ "((std::string)$\\$${left}\\").append(${expression}).append("\\$${right}\\").data()",
    ↳ null, 10),
216 // $"
217 // "
218 (new Regex(@"\\$"""), "\\\"", null, 0),
219 // Console.WriteLine("...")

```

```

220 // printf("...\n")
221 (new Regex(@"Console.WriteLine\("[^"\r\n"]+"")", "printf(\"$1\\n\")", null, 0),
222 // TElement Root;
223 // TElement Root = 0;
224 (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),
225 // TreeElement _elements[N];
226 // TreeElement _elements[N] = { {0} };
227 (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),
228 // auto path = new TElement[MaxPath];
229 // TElement path[MaxPath] = { {0} };
230 (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),
231 // Insert scope borders.
232 // auto added = new StringBuilder();
233 // /*~sb~*/std::string added;
234 (new Regex(@"(auto|(System\\Text\\.)?StringBuilder) (?<variable>[a-zA-Z0-9]+) = new
→ (System\\Text\\.)?StringBuilder\\(\\);"), "/*~$1$2$3$4$5~*/std::string
→ $1$2$3$4$5;", null, 0),
235 // static void Indent(StringBuilder sb, int level)
236 // static void Indent(/*~sb~*/StringBuilder sb, int level)
237 (new Regex(@"(?<start>, \\() (System\\Text\\.)?StringBuilder
→ (?<variable>[a-zA-Z0-9]+) (?<end>, \\))"), "$1$2$3$4$5$6$7$8$9$10$11$12$13$14$15$16$17$18$19$20$21$22$23$24$25$26$27$28$29$30$31$32$33$34$35$36$37$38$39$40$41$42$43$44$45$46$47$48$49$50$51$52$53$54$55$56$57$58$59$60$61$62$63$64$65$66$67$68$69$70$71$72$73$74$75$76$77$78$79$80$81$82$83$84$85$86$87$88$89$90$91$92$93$94$95$96$97$98$99$100$101$102$103$104$105$106$107$108$109$110$111$112$113$114$115$116$117$118$119$120$121$122$123$124$125$126$127$128$129$130$131$132$133$134$135$136$137$138$139$140$141$142$143$144$145$146$147$148$149$150$151$152$153$154$155$156$157$158$159$160$161$162$163$164$165$166$167$168$169$170$171$172$173$174$175$176$177$178$179$180$181$182$183$184$185$186$187$188$189$190$191$192$193$194$195$196$197$198$199$200$201$202$203$204$205$206$207$208$209$210$211$212$213$214$215$216$217$218$219$220$221$222$223$224$225$226$227$228$229$230$231$232$233$234$235$236$237$238$239$240$241$242$243$244$245$246$247$248$249$250$251$252$253$254$255$256$257$258$259$260$261$262$263$264$265$266$267$268$269$270$271$272$273$274$275$276$277$278$279$280$281$282$283$284$285$286$287$288$289$290$291$292$293$294$295$296$297$298$299$300$301$302$303$304$305$306$307$308$309$310$311$312$313$314$315$316$317$318$319$320$321$322$323$324$325$326$327$328$329$330$331$332$333$334$335$336$337$338$339$340$341$342$343$344$345$346$347$348$349$350$351$352$353$354$355$356$357$358$359$360$361$362$363$364$365$366$367$368$369$370$371$372$373$374$375$376$377$378$379$380$381$382$383$384$385$386$387$388$389$390$391$392$393$394$395$396$397$398$399$400$401$402$403$404$405$406$407$408$409$410$411$412$413$414$415$416$417$418$419$420$421$422$423$424$425$426$427$428$429$430$431$432$433$434$435$436$437$438$439$440$441$442$443$444$445$446$447$448$449$450$451$452$453$454$455$456$457$458$459$460$461$462$463$464$465$466$467$468$469$470$471$472$473$474$475$476$477$478$479$480$481$482$483$484$485$486$487$488$489$490$491$492$493$494$495$496$497$498$499$500$501$502$503$504$505$506$507$508$509$510$511$512$513$514$515$516$517$518$519$520$521$522$523$524$525$526$527$528$529$530$531$532$533$534$535$536$537$538$539$540$541$542$543$544$545$546$547$548$549$550$551$552$553$554$555$556$557$558$559$560$561$562$563$564$565$566$567$568$569$570$571$572$573$574$575$576$577$578$579$580$581$582$583$584$585$586$587$588$589$590$591$592$593$594$595$596$597$598$599$600$601$602$603$604$605$606$607$608$609$610$611$612$613$614$615$616$617$618$619$620$621$622$623$624$625$626$627$628$629$630$631$632$633$634$635$636$637$638$639$640$641$642$643$644$645$646$647$648$649$650$651$652$653$654$655$656$657$658$659$660$661$662$663$664$665$666$667$668$669$670$671$672$673$674$675$676$677$678$679$680$681$682$683$684$685$686$687$688$689$690$691$692$693$694$695$696$697$698$699$700$701$702$703$704$705$706$707$708$709$710$711$712$713$714$715$716$717$718$719$720$721$722$723$724$725$726$727$728$729$730$731$732$733$734$735$736$737$738$739$740$741$742$743$744$745$746$747$748$749$750$751$752$753$754$755$756$757$758$759$760$761$762$763$764$765$766$767$768$769$770$771$772$773$774$775$776$777$778$779$780$781$782$783$784$785$786$787$788$789$790$791$792$793$794$795$796$797$798$799$800$801$802$803$804$805$806$807$808$809$810$811$812$813$814$815$816$817$818$819$820$821$822$823$824$825$826$827$828$829$830$831$832$833$834$835$836$837$838$839$840$841$842$843$844$845$846$847$848$849$850$851$852$853$854$855$856$857$858$859$860$861$862$863$864$865$866$867$868$869$870$871$872$873$874$875$876$877$878$879$880$881$882$883$884$885$886$887$888$889$890$891$892$893$894$895$896$897$898$899$900$901$902$903$904$905$906$907$908$909$910$911$912$913$914$915$916$917$918$919$920$921$922$923$924$925$926$927$928$929$930$931$932$933$934$935$936$937$938$939$940$941$942$943$944$945$946$947$948$949$950$951$952$953$954$955$956$957$958$959$960$961$962$963$964$965$966$967$968$969$970$971$972$973$974$975$976$977$978$979$980$981$982$983$984$985$986$987$988$989$990$991$992$993$994$995$996$997$998$999$1000$1001$1002$1003$1004$1005$1006$1007$1008$1009$1010$1011$1012$1013$1014$1015$1016$1017$1018$1019$1020$1021$1022$1023$1024$1025$1026$1027$1028$1029$1030$1031$1032$1033$1034$1035$1036$1037$1038$1039$1040$1041$1042$1043$1044$1045$1046$1047$1048$1049$1050$1051$1052$1053$1054$1055$1056$1057$1058$1059$1060$1061$1062$1063$1064$1065$1066$1067$1068$1069$1070$1071$1072$1073$1074$1075$1076$1077$1078$1079$1080$1081$1082$1083$1084$1085$1086$1087$1088$1089$1090$1091$1092$1093$1094$1095$1096$1097$1098$1099$1100$1101$1102$1103$1104$1105$1106$1107$1108$1109$1110$1111$1112$1113$1114$1115$1116$1117$1118$1119$1120$1121$1122$1123$1124$1125$1126$1127$1128$1129$1130$1131$1132$1133$1134$1135$1136$1137$1138$1139$1140$1141$1142$1143$1144$1145$1146$1147$1148$1149$1150$1151$1152$1153$1154$1155$1156$1157$1158$1159$1160$1161$1162$1163$1164$1165$1166$1167$1168$1169$1170$1171$1172$1173$1174$1175$1176$1177$1178$1179$1180$1181$1182$1183$1184$1185$1186$1187$1188$1189$1190$1191$1192$1193$1194$1195$1196$1197$1198$1199$1200$1201$1202$1203$1204$1205$1206$1207$1208$1209$1210$1211$1212$1213$1214$1215$1216$1217$1218$1219$1220$1221$1222$1223$1224$1225$1226$1227$1228$1229$1230$1231$1232$1233$1234$1235$1236$1237$1238$1239$1240$1241$1242$1243$1244$1245$1246$1247$1248$1249$1250$1251$1252$1253$1254$1255$1256$1257$1258$1259$1260$1261$1262$1263$1264$1265$1266$1267$1268$1269$1270$1271$1272$1273$1274$1275$1276$1277$1278$1279$1280$1281$1282$1283$1284$1285$1286$1287$1288$1289$1290$1291$1292$1293$1294$1295$1296$1297$1298$1299$1300$1301$1302$1303$1304$1305$1306$1307$1308$1309$1310$1311$1312$1313$1314$1315$1316$1317$1318$1319$1320$1321$1322$1323$1324$1325$1326$1327$1328$1329$1330$1331$1332$1333$1334$1335$1336$1337$1338$1339$1340$1341$1342$1343$1344$1345$1346$1347$1348$1349$1350$1351$1352$1353$1354$1355$1356$1357$1358$1359$1360$1361$1362$1363$1364$1365$1366$1367$1368$1369$1370$1371$1372$1373$1374$1375$1376$1377$1378$1379$1380$1381$1382$1383$1384$1385$1386$1387$1388$1389$1390$1391$1392$1393$1394$1395$1396$1397$1398$1399$1400$1401$1402$1403$1404$1405$1406$1407$1408$1409$1410$1411$1412$1413$1414$1415$1416$1417$1418$1419$1420$1421$1422$1423$1424$1425$1426$1427$1428$1429$1430$1431$1432$1433$1434$1435$1436$1437$1438$1439$1440$1441$1442$1443$1444$1445$1446$1447$1448$1449$1450$1451$1452$1453$1454$1455$1456$1457$1458$1459$1460$1461$1462$1463$1464$1465$1466$1467$1468$1469$1470$1471$1472$1473$1474$1475$1476$1477$1478$1479$1480$1481$1482$1483$1484$1485$1486$1487$1488$1489$1490$1491$1492$1493$1494$1495$1496$1497$1498$1499$1500$1501$1502$1503$1504$1505$1506$1507$1508$1509$1510$1511$1512$1513$1514$1515$1516$1517$1518$1519$1520$1521$1522$1523$1524$1525$1526$1527$1528$1529$1530$1531$1532$1533$1534$1535$1536$1537$1538$1539$1540$1541$1542$1543$1544$1545$1546$1547$1548$1549$1550$1551$1552$1553$1554$1555$1556$1557$1558$1559$1560$1561$1562$1563$1564$1565$1566$1567$1568$1569$1570$1571$1572$1573$1574$1575$1576$1577$1578$1579$1580$1581$1582$1583$1584$1585$1586$1587$1588$1589$1590$1591$1592$1593$1594$1595$1596$1597$1598$1599$1600$1601$1602$1603$1604$1605$1606$1607$1608$1609$1610$1611$1612$1613$1614$1615$1616$1617$1618$1619$1620$1621$1622$1623$1624$1625$1626$1627$1628$1629$1630$1631$1632$1633$1634$1635$1636$1637$1638$1639$1640$1641$1642$1643$1644$1645$1646$1647$1648$1649$1650$1651$1652$1653$1654$1655$1656$1657$1658$1659$1660$1661$1662$1663$1664$1665$1666$1667$1668$1669$1670$1671$1672$1673$1674$1675$1676$1677$1678$1679$1680$1681$1682$1683$1684$1685$1686$1687$1688$1689$1690$1691$1692$1693$1694$1695$1696$1697$1698$1699$1700$1701$1702$1703$1704$1705$1706$1707$1708$1709$1710$1711$1712$1713$1714$1715$1716$1717$1718$1719$1720$1721$1722$1723$1724$1725$1726$1727$1728$1729$1730$1731$1732$1733$1734$1735$1736$1737$1738$1739$1740$1741$1742$1743$1744$1745$1746$1747$1748$1749$1750$1751$1752$1753$1754$1755$1756$1757$1758$1759$1760$1761$1762$1763$1764$1765$1766$1767$1768$1769$1770$1771$1772$1773$1774$1775$1776$1777$1778$1779$1780$1781$1782$1783$1784$1785$1786$1787$1788$1789$1790$1791$1792$1793$1794$1795$1796$1797$1798$1799$1800$1801$1802$1803$1804$1805$1806$1807$1808$1809$1810$1811$1812$1813$1814$1815$1816$1817$1818$1819$1820$1821$1822$1823$1824$1825$1826$1827$1828$1829$1830$1831$1832$1833$1834$1835$1836$1837$1838$1839$1840$1841$1842$1843$1844$1845$1846$1847$1848$1849$1850$1851$1852$1853$1854$1855$1856$1857$1858$1859$1860$1861$1862$1863$1864$1865$1866$1867$1868$1869$1870$1871$1872$1873$1874$1875$1876$1877$1878$1879$1880$1881$1882$1883$1884$1885$1886$1887$1888$1889$1890$1891$1892$1893$1894$1895$1896$1897$1898$1899$1900$1901$1902$1903$1904$1905$1906$1907$1908$1909$1910$1911$1912$1913$1914$1915$1916$1917$1918$1919$1920$1921$1922$1923$1924$1925$1926$1927$1928$1929$1930$1931$1932$1933$1934$1935$1936$1937$1938$1939$1940$1941$1942$1943$1944$1945$1946$1947$1948$1949$1950$1951$1952$1953$1954$1955$1956$1957$1958$1959$1960$1961$1962$1963$1964$1965$1966$1967$1968$1969$1970$1971$1972$1973$1974$1975$1976$1977$1978$1979$1980$1981$1982$1983$1984$1985$1986$1987$1988$1989$1990$1991$1992$1993$1994$1995$1996$1997$1998$1999$2000$2001$2002$2003$2004$2005$2006$2007$2008$2009$2010$2011$2012$2013$2014$2015$2016$2017$2018$2019$2020$2021$2022$2023$2024$2025$2026$2027$2028$2029$2030$2031$2032$2033$2034$2035$2036$2037$2038$2039$2040$2041$2042$2043$2044$2045$2046$2047$2048$2049$2050$2051$2052$2053$2054$2055$2056$2057$2058$2059$2060$2061$2062$2063$2064$2065$2066$2067$2068$2069$2070$2071$2072$2073$2074$2075$2076$2077$2078$2079$2080$2081$2082$2083$2084$2085$2086$2087$2088$2089$2090$2091$2092$2093$2094$2095$2096$2097$2098$2099$2100$2101$2102$2103$2104$2105$2106$2107$2108$2109$2110$2111$2112$2113$2114$2115$2116$2117$2118$2119$2120$2121$2122$2123$2124$2125$2126$2127$2128$2129$2130$2131$2132$2133$2134$2135$2136$2137$2138$2139$2140$2141$2142$2143$2144$2145$2146$2147$2148$2149$2150$2151$2152$2153$2154$2155$2156$2157$2158$2159$2160$2161$2162$2163$2164$2165$2166$2167$2168$2169$2170$2171$2172$2173$2174$2175$2176$2177$2178$2179$2180$2181$2182$2183$2184$2185$2186$2187$2188$2189$2190$2191$2192$2193$2194$2195$2196$2197$2198$2199$2200$2201$2202$2203$2204$2205$2206$2207$2208$2209$2210$2211$2212$2213$2214$2215$2216$2217$2218$2219$2220$2221$2222$2223$2224$2225$2226$2227$2228$2229$2230$2231$2232$2233$2234$2235$2236$2237$2238$2239$2240$2241$2242$2243$2244$2245$2246$2247$2248$2249$2250$2251$2252$2253$2254$2255$2256$2257$2258$2259$2260$2261$2262$2263$2264$2265$2266$2267$2268$2269$2270$2271$2272$2273$2274$2275$2276$2277$2278$2279$2280$2281$2282$2283$2284$2285$2286$2287$2288$2289$2290$2291$2292$2293$2294$2295$2296$2297$2298$2299$2300$2301$2302$2303$2304$2305$2306$2307$2308$2309$2310$2311$2312$2313$2314$2315$2316$2317$2318$2319$2320$2321$2322$2323$2324$2325$2326$2327$2328$2329$2330$2331$2332$2333$2334$2335$2336$2337$2338$2339$2340$2341$2342$2343$2344$2345$2346$2347$2348$2349$2350$2351$2352$2353$2354$2355$2356$2357$2358$2359$2360$2361$2362$2363$2364$2365$2366$2367$2368$2369$2370$2371$2372$2373$2374$2375$2376$2377$2378$2379$2380$2381$2382$2383$2384$2385$2386$2387$2388$2389$2390$2391$2392$2393$2394$2395$2396$2397$2398$2399$2400$2401$2402$2403$2404$2405$2406$2407$2408$2409$2410$2411$2412$2413$2414$2415$2416$2417$2418$2419$2420$2421$2422$2423$2424$2425$2426$2427$2428$2429$2430$2431$2432$2433$2434$2435$2436$2437$2438$2439$2440$2441$2442$2443$2444$2445$2446$2447$2448$2449$2450$2451$2452$2453$2454$2455$2456$2457$2458$2459$2460$2461$2462$2463$2464$2465$2466$2467$2468$2469$2470$2471$2472$2473$2474$2475$2476$2477$2478$2479$2480$2481$2482$2483$2484$2485$2486$2487$2488$2489$2490$2491$2492$2493$2494$2495$2496$2497$2498$2499$2500$2501$2502$2503$2504$2505$2506$2507$2508$2509$2510$2511$2512$2513$2514$2515$2516$2517$2518$2519$2520$2521$2522$2523$2524$2525$2526$2527$2528$2529$2530$2531$2532$2533$2534$2535$2536$2537$2538$2539$2540$2541$2542$2543$2544$2545$2546$2547$2548$2549$2550$2551$2552$2553$2554$2555$2556$2557$2558$2559$2560$2561$2562$2563$2564$2565$2566$2567$2568$2569$2570$2571$2572$2573$2574$2575$25
```

```

(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();
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),

```

```

315 }.Cast<ISubstitutionRule>().ToList();
316
317 public static readonly IList<ISubstitutionRule> LastStage = new List<SubstitutionRule>
318 {
319     // ICounter<int, int> c1;
320     // ICounter<int, int>* c1;
321     (new Regex(@"(?<abstractType>I[A-Z][a-zA-Z0-9]+(<[^\r\n]+>)?
    ↳ (?<variable>[_a-zA-Z0-9]+);"), "${abstractType}* ${variable}";", null, 0),
322     // (expression)
323     // expression
324     (new Regex(@"(\(|\)|)(([a-zA-Z0-9_\*:]+)\(|\)|;|\\|)"), "$1$2$3", null, 0),
325     // (method(expression))
326     // method(expression)
327     (new Regex(@"(?<firstSeparator>(\(|
    ↳ ))\((?<method>[a-zA-Z0-9_\*:]+)\((?<expression>((?<parenthesis>\(|(?<-parenthesis>\)|[a-zA-Z0-9_\*:]+)
    ↳ (?<parenthesis>(?!))\)|(?<lastSeparator>(\(|;|\\|))"), "${firstSeparator}${method}(${expression})${lastSeparator}";", null, 0),
328     // return ref _elements[node];
329     // return &_elements[node];
330     (new Regex(@"return ref ([_a-zA-Z0-9]+)\([[_a-zA-Z0-9_\*:]+\];");", "return &$1[$2];",
    ↳ null, 0),
331     // null
332     // NULL
333     (new Regex(@"(?<before>\r?\n[^\r\n]*(\"\\\"|\"[^\r\n]*\"[^\r\n]*)(?<=\\W)null
    ↳ (?<after>\\W)");", "${before}NULL${after}";", null,
    ↳ 10),
334     // default
335     // 0
336     (new Regex(@"(?<before>\r?\n[^\r\n]*(\"\\\"|\"[^\r\n]*\"[^\r\n]*)(?<=\\W)defa
    ↳ ult(?<after>\\W)");", "${before}0${after}";", null,
    ↳ 10),
337     // #region Always
338     //
339     (new Regex(@"(^\r?\n)[ \t]*#(region|endregion)[^\r\n]*(\r?\n|$)");", "", null, 0),
340     // //define ENABLE_TREE_AUTO_DEBUG_AND_VALIDATION
341     //
342     (new Regex(@"\\\/[ \t]*#define[ \t]+[_a-zA-Z0-9]+[ \t]*");", "", null, 0),
343     // #if USEARRAYPOOL\r\n#endif
344     //
345     (new Regex(@"#if [a-zA-Z0-9]+\s+#endif");", "", null, 0),
346     // [Fact]
347     //
348     (new Regex(@"(?<firstNewLine>\r?\n|\A)(?<indent>[ \t
    ↳ ]+)\[([a-zA-Z0-9]+(\((?<expression>((?<parenthesis>\(|(?<-parenthesis>\)|[^\r\n]*)+)?(parenthesis)(?!))\)|[^\r\n]*)+)?\]
    ↳ [ \t]*\(\r?\n\k<indent>?\)");", "${firstNewLine}${indent}";", null, 5),
349     // \n ... namespace
350     // namespace
351     (new Regex(@"(\\S[\\r\\n]{1,2})?[\\r\\n]+namespace");", "$1namespace", null, 0),
352     // \n ... class
353     // class
354     (new Regex(@"(\\S[\\r\\n]{1,2})?[\\r\\n]+class");", "$1class", null, 0),
355 }.Cast<ISubstitutionRule>().ToList();
356
357 public CSharpToCppTransformer(IList<ISubstitutionRule> extraRules) :
    ↳ base(FirstStage.Concat(extraRules).Concat(LastStage).ToList()) { }
358
359 public CSharpToCppTransformer() : base(FirstStage.Concat(LastStage).ToList()) { }
360 }
361 }

```

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

```

1 using Xunit;
2
3 namespace Platform.RegularExpressions.Transformer.CSharpToCpp.Tests
4 {
5     public class CSharpToCppTransformerTests
6     {
7         [Fact]
8         public void EmptyLineTest()
9         {
10             // This test can help to test basic problems with regular expressions like incorrect
            ↳ syntax
11             var transformer = new CSharpToCppTransformer();
12             var actualResult = transformer.Transform("", new Context(null));
13             Assert.Equal("", actualResult);
14         }
15     }

```

```
16         [Fact]
17         public void HelloWorldTest()
18         {
19             const string helloWorldCode = @"using System;
20 class Program
21 {
22     public static void Main(string[] args)
23     {
24         Console.WriteLine(""Hello, world!"");
25     }
26 }";
27             const string expectedResult = @"class Program
28 {
29     public: static void Main(const char* args[])
30     {
31         printf(""Hello, world!\n"");
32     }
33 };";
34             var transformer = new CSharpToCppTransformer();
35             var actualResult = transformer.Transform(helloWorldCode, new Context(null));
36             Assert.Equal(expectedResult, actualResult);
37         }
38     }
39 }
```


Index

./Platform.RegularExpressions.Transformer.CSharpToCpp.Tests/CSharpToCppTransformerTests.cs, 7
./Platform.RegularExpressions.Transformer.CSharpToCpp/CSharpToCppTransformer.cs, 1