

LinksPlatform's Platform.RegularExpressions.Transformer.CSharpToCpp Class Library

./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
```

```

64 (new Regex(@"(protected|private) readonly ([a-zA-Z<>0-9]+)([\\\[\\]]+)
    ↪ ([_a-zA-Z0-9]+);"), "$2 $4[N]";", null, 0),
65 // protected readonly TElement Zero;
66 // TElement Zero;
67 (new Regex(@"(protected|private) readonly ([a-zA-Z<>0-9]+) ([_a-zA-Z0-9]+);"), "$2
    ↪ $3";", null, 0),
68 // private
69 //
70 (new Regex(@"(\W)(private|protected|public|internal) "), "$1", null, 0),
71 // SizeBalancedTree(int capacity) => a = b;
72 // SizeBalancedTree(int capacity) { a = b; }
73 (new Regex(@"(^s+)(override )?(void )?([a-zA-Z0-9]+)\(((^\\([+])\\)s+=>s+([~;]+);"),
    ↪ "$1$2$3$4($5) { $6; }", null, 0),
74 // () => Integer<TElement>.Zero,
75 // () { return Integer<TElement>.Zero; },
76 (new Regex(@"\\(\\)s+=>s+([~\\r\\n;]+?);"), "()" { return $1; }", null, 0),
77 // => Integer<TElement>.Zero;
78 // { return Integer<TElement>.Zero; }
79 (new Regex(@"\\(\\)s+=>s+([~\\r\\n;]+?);"), "()" { return $1; }", null, 0),
80 // () { return avlTree.Count; }
81 // [&]()-> auto { return avlTree.Count; }
82 (new Regex(@"", "\\(\\) { return ([~;]+); }"), " [&]()-> auto { return $1; }", null, 0),
83 // Count => GetSizeOrZero(Root);
84 // GetCount() { return GetSizeOrZero(Root); }
85 (new Regex(@"([A-Z][a-z]+)s+=>s+([~;]+);"), "Get$1() { return $2; }", null, 0),
86 // var
87 // auto
88 (new Regex(@"(\W)var(\W)"), "$1auto$2", null, 0),
89 // unchecked
90 //
91 (new Regex(@"[\\r\\n]{2}s*?unchecked\s*?$"), "", null, 0),
92 // $"
93 // "
94 (new Regex(@"\$"""), "\\\"", null, 0),
95 // Console.WriteLine("...")
96 // printf("...\n")
97 (new Regex(@"Console\\.WriteLine\\(\"\"([~\""]+)[~\""]\\)\""), "printf(\"$1\\n\")", null, 0),
98 // throw new InvalidOperationException
99 // throw std::exception
100 (new Regex(@"throw new (InvalidOperationException|Exception)", "throw
    ↪ std::exception", null, 0),
101 // override void PrintNode(TElement node, StringBuilder sb, int level)
102 // void PrintNode(TElement node, StringBuilder sb, int level) override
103 (new Regex(@"override ([a-zA-Z0-9 \\*+]+)(\\([~\\)]+?\\))"), "$1$2 override", null, 0),
104 // string
105 // char*
106 (new Regex(@"(\W)string(\W)"), "$1char*$2", null, 0),
107 // sbyte
108 // std::int8_t
109 (new Regex(@"(\W)sbyte(\W)"), "$1std::int8_t$2", null, 0),
110 // uint
111 // std::uint32_t
112 (new Regex(@"(\W)uint(\W)"), "$1std::uint32_t$2", null, 0),
113 // char*[] args
114 // char* args[]
115 (new Regex(@"([_a-zA-Z0-9:~*+]?)\\(\\ ([a-zA-Z0-9]+)"), "$1 $2[]", null, 0),
116 // using Platform.Numbers;
117 //
118 (new Regex(@"([\\r\\n]{2}|^~)s*?using [\\.a-zA-Z0-9]+;s*?$"), "", null, 0),
119 // struct TreeElement { }
120 // struct TreeElement { };
121 (new Regex(@"(struct|class) ([a-zA-Z0-9]+)(\\s+){([\\sa-zA-Z0-9;:_]+?)}([~;])"), "$1
    ↪ $2$3{$4};$5", null, 0),
122 // class Program { }
123 // class Program { };
124 (new Regex(@"(struct|class) ([a-zA-Z0-9]+[~\\r\\n]*)([\\r\\n]+(?<indentLevel>[\\t
    ↪ ]*))?\\{([\\S\\s]?[\\r\\n]+\\k<indentLevel>)\\}([~;]|$)", "$1 $2$3{$4};$5", null, 0),
125 // class SizedBinaryTreeMethodsBase : GenericCollectionMethodsBase
126 // class SizedBinaryTreeMethodsBase : public GenericCollectionMethodsBase
127 (new Regex(@"class ([a-zA-Z0-9]+) : ([a-zA-Z0-9]+)"), "class $1 : public $2", null,
    ↪ 0),
128 // Insert scope borders.
129 // ref TElement root
130 // ~!root!~ref TElement root
131 (new Regex(@"(?<definition>(?!<= |\\() (ref [a-zA-Z0-9]+|[a-zA-Z0-9]+(?<!ref))
    ↪ (?<variable>[a-zA-Z0-9]+)(?=\\)|, | =)")), "~!${variable}!~${definition}", null,
    ↪ 0),

```

```

132 // Inside the scope of ~!root!~ replace:
133 // root
134 // *root
135 (new Regexp(@"(?<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 Regexp(@"~!(?<pointer>[a-zA-Z0-9]+)!~"), "", null, 5),
140 // ref auto root = ref
141 // ref auto root =
142 (new Regexp(@"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 Regexp(@"\\*([a-zA-Z0-9]+) = ref ([a-zA-Z0-9]+)(\\W)"), "$1 = $2$3", null, 0),
146 // (ref left)
147 // (left)
148 (new Regexp(@"\\(ref ([a-zA-Z0-9]+)(\\|\\(|,))"), "($1$2", null, 0),
149 // ref TElement
150 // TElement*
151 (new Regexp(@"(\\|\\()ref ([a-zA-Z0-9]+) "), "$1$2* ", null, 0),
152 // ref sizeBalancedTree2.Root
153 // &sizeBalancedTree2->Root
154 (new Regexp(@"ref ([a-zA-Z0-9]+)\\.([a-zA-Z0-9\\*]+)"), "&$1->$2", null, 0),
155 // ref GetElement(node).Right
156 // &GetElement(node)->Right
157 (new Regexp(@"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 Regexp(@"([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 Regexp(@"\\[Fact\\] \\s\\n\\n+(static )?void ([a-zA-Z0-9]+)\\(\\)"), "TEST_METHOD($2)",
→ null, 0),
164 // class TreesTests
165 // TEST_CLASS(TreesTests)
166 (new Regexp(@"class ([a-zA-Z0-9]+)Tests"), "TEST_CLASS($1)", null, 0),
167 // Assert.Equal
168 // Assert::AreEqual
169 (new Regexp(@"Assert\\.Equal"), "Assert::AreEqual", null, 0),
170 // TElement Root;
171 // TElement Root = 0;
172 (new Regexp(@"(\\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 Regexp(@"(\\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 Regexp(@"(\\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 scope = new HashSet<TElement>();
181 // ~!added!~std::unordered_set<TElement> added;
182 (new Regexp(@"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 Regexp(@"(?<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 Regexp(@"(?<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)) {

```

```

192 // if (!added.contains(node)) { added.insert(node);
193 (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),
194 // Remove scope borders.
195 // ~!added!~
196 //
197 (new Regex(@"~!(?<pointer>[a-zA-Z0-9]+)!~"), "", null, 5),
198 // Insert scope borders.
199 // auto random = new System.Random(0);
200 // std::srand(0);
201 (new Regex(@"[a-zA-Z0-9\.]+ ([a-zA-Z0-9]+) = new
    ↳ (System\.)?Random\((([a-zA-Z0-9]+\)\);", "~!$1!~std::srand($3);", null, 0),
202 // Inside the scope of ~!random!~ replace:
203 // random.Next(1, N)
204 // (std::rand() % N) + 1
205 (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),
206 // Remove scope borders.
207 // ~!random!~
208 //
209 (new Regex(@"~!(?<pointer>[a-zA-Z0-9]+)!~"), "", null, 5),
210 }.Cast<ISubstitutionRule>().ToList();
211
212 public static readonly IList<ISubstitutionRule> LastStage = new List<SubstitutionRule>
213 {
214     // (expression)
215     // expression
216     (new Regex(@"\((| )\((([a-zA-Z0-9_\*:]+)\)(,| |;|\\))", "$1$2$3", null, 0),
217     // method(expression)
218     // method(expression)
219     (new Regex(@"(?<firstSeparator>\((|
    ↳ ))\(((?<method>[a-zA-Z0-9_\->*\:]+)\(((?<expression>((?<parenthesis>\( )|(?<-parenthesis>
    ↳ hesis>\\))|([a-zA-Z0-9_\->*\:]*+)(?(parenthesis)(?!))\\)\)(?<lastSeparator>(,|
    ↳ |;|\\))") , "${firstSeparator}${method}(${expression})${lastSeparator}", null, 0),
220 // return ref _elements[node];
221 // return &_elements[node];
222 (new Regex(@"return ref ([_a-zA-Z0-9]+\)\((([_a-zA-Z0-9_\*:]+)\);", "return &$1[$2];",
    ↳ null, 0),
223 // default
224 // 0
225 (new Regex(@"(\\W)default(\\W)", "${1}0$2", null, 0),
226 // // #define ENABLE_TREE_AUTO_DEBUG_AND_VALIDATION
227 //
228 (new Regex(@"\\\/\\\/[ \t]*#define[ \t]+[_a-zA-Z0-9]+[ \t]*") , "", null, 0),
229 // #if USEARRAYPOOL\r\n#endif
230 //
231 (new Regex(@"#if [a-zA-Z0-9]+\s+#endif") , "", null, 0),
232 // [Fact]
233 //
234 (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),
235 // \n ... namespace
236 // namespace
237 (new Regex(@"(\\S[\\r\\n]{1,2})?[\\r\\n]+namespace") , "$1namespace", null, 0),
238 // \n ... class
239 // class
240 (new Regex(@"(\\S[\\r\\n]{1,2})?[\\r\\n]+class") , "$1class", null, 0),
241 }.Cast<ISubstitutionRule>().ToList();
242
243 public CSharpToCppTransformer(IList<ISubstitutionRule> extraRules) :
    ↳ base(FirstStage.Concat(extraRules).Concat(LastStage).ToList()) { }
244
245 public CSharpToCppTransformer() : base(FirstStage.Concat(LastStage).ToList()) { }
246 }
247 }

```

./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 HelloWorldTest()
9         {
10             const string helloWorldCode = @"using System;
11 class Program
12 {
13     public static void Main(string[] args)
14     {
15         Console.WriteLine("Hello, world!");
16     }
17 }";
18             const string expectedResult = @"class Program
19 {
20     public:
21     static void Main(char* args[])
22     {
23         printf("Hello, world!\n");
24     }
25 };";
26             var transformer = new CSharpToCppTransformer();
27             var actualResult = transformer.Transform(helloWorldCode, new Context(null));
28             Assert.Equal(expectedResult, actualResult);
29         }
30     }
31 }
```

Index

./Platform.RegularExpressions.Transformer.CSharpToCpp.Tests/CSharpToCppTransformerTests.cs, 4
./Platform.RegularExpressions.Transformer.CSharpToCpp/CSharpToCppTransformer.cs, 1