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

## 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 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]+)>\(([^\)\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),
            // (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),
            // Predicate<TArgument> predicate
            // std::function<bool(TArgument)> predicate
            (new Regex(@"Predicate<([a-zA-Z0-9]+)> ([a-zA-Z0-9]+)"), "std::function<bool($1)> $2", null, 0),
            // public static readonly EnsureAlwaysExtensionRoot Always = new EnsureAlwaysExtensionRoot();
            // inline static EnsureAlwaysExtensionRoot Always;
            (new Regex(@"public static readonly (?<type>[a-zA-Z0-9]+) (?<name>[a-zA-Z0-9_]+) = new \k<type>\(\);"), "inline static ${type} ${name};", null, 0),
            // public static readonly string ExceptionContentsSeparator = "---";
            // inline static const char* ExceptionContentsSeparator = "---";
            (new Regex(@"public static readonly string (?<name>[a-zA-Z0-9_]+) = ""(?<string>(\""|[^""\r\n])+)"";"), "inline static const char* ${name} = \"${string}\";", null, 0),
            // private const int MaxPath = 92;
            // static const int MaxPath = 92;
            (new Regex(@"private (const|static readonly) ([a-zA-Z0-9]+) ([_a-zA-Z0-9]+) = ([^;\r\n]+);"), "static const $2 $3 = $4;", 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 virtual
            // virtual
            (new Regex(@"protected virtual"), "virtual", null, 0),
            // protected abstract TElement GetFirst();
            // virtual TElement GetFirst() = 0;
            (new Regex(@"protected abstract ([^;\r\n]+);"), "virtual $1 = 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),
            // 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),
            // static void NotImplementedException(ThrowExtensionRoot root) => throw new NotImplementedException();
            // static void NotImplementedException(ThrowExtensionRoot root) { return throw new NotImplementedException(); }
            (new Regex(@"(^\s+)(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) { throw$8; }", null, 0),
            // SizeBalancedTree(int capacity) => a = b;
            // SizeBalancedTree(int capacity) { a = b; }
            (new Regex(@"(^\s+)(template \<[^>\r\n]+\> )?(static )?(override )?(void )?([a-zA-Z0-9]+)\(([^\(\r\n]*)\)\s+=>\s+([^;\r\n]+);"), "$1$2$3$4$5$6($7) { $8; }", null, 0),
            // int SizeBalancedTree(int capacity) => a;
            // int SizeBalancedTree(int capacity) { return a; }
            (new Regex(@"(^\s+)(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) { return $8; }", 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 ([^;\r\n]+); }"), ", [&]()-> auto { return $1; }", null, 0),
            // Count => GetSizeOrZero(Root);
            // GetCount() { return GetSizeOrZero(Root); }
            (new Regex(@"([A-Z][a-z]+)\s+=>\s+([^;\r\n]+);"), "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\(""([^""\r\n]+)""\)"), "printf(\"$1\\n\")", null, 0),
            // throw new InvalidOperationException
            // throw std::runtime_error
            (new Regex(@"throw new (InvalidOperationException|Exception)"), "throw std::runtime_error", null, 0),
            // void RaiseExceptionIgnoredEvent(Exception exception)
            // void RaiseExceptionIgnoredEvent(const std::exception& exception)
            (new Regex(@"(\(|, )(System\.Exception|Exception)( |\))"), "$1const std::exception&$3", null, 0),
            // EventHandler<Exception>
            // EventHandler<std::exception>
            (new Regex(@"(\W)(System\.Exception|Exception)(\W)"), "$1std::exception$3", 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 \*\+]+)(\([^\)\r\n]+?\))"), "$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),
            // @object
            // object
            (new Regex(@"@([_a-zA-Z0-9]+)"), "$1", null, 0),
            // using Platform.Numbers;
            //
            (new Regex(@"([\r\n]{2}|^)\s*?using [\.a-zA-Z0-9]+;\s*?$"), "", 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 $2", null, 0),
            // class IProperty : ISetter<TValue, TObject>, IProvider<TValue, TObject>
            // class IProperty : public ISetter<TValue, TObject>, IProvider<TValue, TObject>
            (new Regex(@"(?<before>class [a-zA-Z0-9]+ : ((public [a-zA-Z0-9]+(<[a-zA-Z0-9 ,]+>)?, )+)?)(?<inheritedType>(?!public)[a-zA-Z0-9]+(<[a-zA-Z0-9 ,]+>)?)(?<after>(, [a-zA-Z0-9]+(?!>)|[ \r\n]+))"), "${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()
            // TEST_METHOD(SizeBalancedTreeMultipleAttachAndDetachTest)
            (new Regex(@"\[Fact\][\s\n]+(static )?void ([a-zA-Z0-9]+)\(\)"), "TEST_METHOD($2)", 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),
            // TElement Root;
            // TElement Root = 0;
            (new Regex(@"(\r?\n[\t ]+)([a-zA-Z0-9:_]+(?<!return)) ([_a-zA-Z0-9]+);"), "$1$2 $3 = 0;", null, 0),
            // TreeElement _elements[N];
            // TreeElement _elements[N] = { {0} };
            (new Regex(@"(\r?\n[\t ]+)([a-zA-Z0-9]+) ([_a-zA-Z0-9]+)\[([_a-zA-Z0-9]+)\];"), "$1$2 $3[$4] = { {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 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);
            (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),
            // Insert method body scope starts.
            // void PrintNodes(TElement node, StringBuilder sb, int level) {
            // void PrintNodes(TElement node, StringBuilder sb, int level) {/*method-start*/
            (new Regex(@"(?<start>\r?\n[\t ]+)(?<prefix>((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*/${end}", null, 0),
            // Insert method body scope ends.
            // {/*method-start*/...}
            // {/*method-start*/.../*method-end*/}
            (new Regex(@"\{/\*method-start\*/(?<body>((?<bracket>\{)|(?<-bracket>\})|[^\{\}]*)+)\}"), "{/*method-start*/${body}/*method-end*/}", null, 0),
            // Inside method bodies replace:
            // GetFirst(
            // this->GetFirst(
            //(new Regex(@"(?<separator>(\(|, |([\W]) |return ))(?<!(->|\* ))(?<method>(?!sizeof)[a-zA-Z0-9]+)\((?!\) \{)"), "${separator}this->${method}(", null, 1),
            (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),
            // Remove scope borders.
            // /*method-start*/
            //
            (new Regex(@"/\*method-(start|end)\*/"), "", null, 0),
            // throw new ArgumentNullException(argumentName, message);
            // throw std::invalid_argument(((std::string)"Argument ").append(argumentName).append(" is null: ").append(message).append("."));
            (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),
            // throw new ArgumentException(message, argumentName);
            // throw std::invalid_argument(((std::string)"Invalid ").append(argumentName).append(" argument: ").append(message).append("."));
            (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),
            // throw new NotSupportedException();
            // throw std::logic_error("Not supported exception.");
            (new Regex(@"throw new NotSupportedException\(\);"), "throw std::logic_error(\"Not supported exception.\");", null, 0),
            // throw new NotImplementedException();
            // throw std::logic_error("Not implemented exception.");
            (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>\()|(?<-parenthesis>\))|[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>\()|(?<-parenthesis>\))|[^()\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 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