CODE SPITZ 82

# KOTLIN ELEMENTARY

1 2 [✓ 3] 4 5 6

## Null & Nullable

```kotlin
val double:(Int)->Int = {it * 2}
val double:(Int)->Int = {it * 2}

val v0:Int = 3
println("${double(v0)}")
```

```
val double:(Int)->Int = {it * 2}

val v0:Int = null compile error!
println("${double(v0)}")
```

```
val double:(Int)->Int = {it * 2}

val v0:Int = 3
println("${double(v0)}")

val v1:Int? = null
val double:(Int)->Int = {it * 2}

val v0:Int = 3
println("${double(v0)}")

val v1:Int? = null
println("${double(v1)}") compile error!
```

```kotlin
val double:(Int)->Int = {it * 2}

val v0:Int = 3
println("${double(v0)}")

val v1:Int? = null
if(v1 != null) println("${double(v1)}")
```

```kotlin
val double:(Int)->Int = {it * 2}

val v0:Int = 3
println("${double(v0)}")

val v1:Int? = null
if(v1 != null) println("${double(v1)}")
```

Int? → Int

smart cast

## inline function

# before Inline

```
fun pass(v:Int, block:(Int)->Int) = block(v)
```

## before Inline

```
fun pass(v:Int, block:(Int)->Int) = block(v)
```

```
println("${pass(3){it * 2}}")
```

## before Inline

```kotlin
fun pass(v:Int, block:(Int)->Int) = block(v)
```

```kotlin
println("${pass(3){it * 2}}")
```

```javascript
function main$lambda(it) {
  return it * 2 | 0;
}
function pass(v, block) {
  return block(v);
}
println(pass(3, main$lambda).toString());
```

# after Inline

```kotlin
inline fun pass(v:Int, block:(Int)->Int) = block(v)
```

```kotlin
println("${pass(3){it * 2}}")
```

## after Inline

```
inline fun pass(v:Int, block:(Int)->Int) = block(v)
```

```
println("${pass(3){it * 2}}")
```

```
println((3 * 2 | 0).toString());
```

## if

```kotlin
inline fun ifTrue(v:Boolean, block:()->Unit){if(v) block()}
```

## if

```kotlin
inline fun ifTrue(v:Boolean, block:()->Unit){if(v) block()}
```

```kotlin
ifTrue(true){
    println("true")
}
```

## if

```
inline fun ifTrue(v:Boolean, block:()->Unit){if(v) block()}
```

```
ifTrue(true){
    println("true")
}
```

```
if (true) {
  println('true');
}
```

## reverseFor

```
inline fun <T>reverseFor(v:List<T>, block: (T) -> Unit){
    var i = v.size
    while(i-- > 0) block(v[i])
}
```

# reverseFor

```kotlin
inline fun <T>reverseFor(v:List<T>, block: (T) -> Unit){
    var i = v.size
    while(i-- > 0) block(v[i])
}
```

```kotlin
reverseFor( listOf("a", "b", "c"), ::println)
```

## reverseFor

```kotlin
inline fun <T>reverseFor(v:List<T>, block: (T) -> Unit){
    var i = v.size
    while(i-- > 0) block(v[i])
}
```

```kotlin
reverseFor( listOf("a", "b", "c"), ::println)
```

Function reference operator

# reverseFor

```
inline fun <T>reverseFor(v:List<T>, block: (T) -> Unit){
    var i = v.size
    while(i-- > 0) block(v[i])
}
```

```
reverseFor( listOf("a", "b", "c"), ::println)
```

```
var v = listOf(['a', 'b', 'c']);
var tmp$;
var i = v.size;
while ((tmp$ = i, i = tmp$ - 1 | 0, tmp$) > 0) {
  println(v.get_za3lpa$(i));
}
```

## Extention function (extensions)

## trim

```
"  aaa  ".trim()
```

## trim

```kotlin
"    aaa    ".trim()
```

```kotlin
inline fun String.trim(): String = (this as CharSequence).trim().toString()
```

receiver

# trim

```
"   aaa   ".trim()
```

```
inline fun String.trim(): String = (this as CharSequence).trim().toString()
```

# trim

```
"   aaa   ".trim()
```

```
inline fun String.trim(): String = (this as CharSequence).trim().toString()
```

```
var trim = Kotlin.kotlin.text.trim_gw00vp$;
var tmp$_0;
trim(Kotlin.isCharSequence(tmp$_0 = '   aaa   ') ? tmp$_0 : throwCCE()).toString();
```

# trim

```
"   aaa   ".trim()
```

```
inline fun String.trim(): String = (this as CharSequence).trim().toString()
```

```
var trim = Kotlin.kotlin.text.trim_gw00vp$;
var tmp$_0;
trim(Kotlin.isCharSequence(tmp$_0 = '   aaa   ') ? tmp$_0 : throwCCE()).toString();
```

receiver

# pop

```kotlin
fun <T> MutableList<T>.pop() = if(isEmpty()) null else removeAt(lastIndex)
```

```kotlin
val list = mutableListOf("a", "b", "c")
val last = list.pop()
println("last = $last, list = [${list.joinToString(",")}]")
```

```javascript
function pop($receiver) {
  return $receiver.isEmpty() ? null : $receiver.removeAt_za3lpa$(get_lastIndex($receiver));
}
var list = mutableListOf(['a', 'b', 'c']);
var last = pop(list);
println('last = ' + toString(last) + ', list = [' + joinToString(list, ',') + ']');
```

# pop

```
fun <T> MutableList<T>.pop() = if(isEmpty()) null else removeAt(lastIndex)
```

```
val list = mutableListOf("a", "b", "c")
val last = list.pop()
println("last = $last, list = [${list.joinToString(",")}]")
```

```
function pop($receiver) {
  return $receiver.isEmpty() ? null : $receiver.removeAt_za3lpa$(get_lastIndex($receiver));
}
var list = mutableListOf(['a', 'b', 'c']);
var last = pop(list);
println('last = ' + toString(last) + ', list = [' + joinToString(list, ',') + ']');
```

## kotlin inline function

# kotlin inline functions

```
inline fun TODO(): Nothing
inline fun TODO(reason: String): Nothing
inline fun <R> run(block: () -> R): R
inline fun <T, R> T.run(block: T.() -> R): R
inline fun <T, R> with(receiver: T, block: T.() -> R): R
inline fun <T> T.apply(block: T.() -> Unit): T
inline fun <T> T.also(block: (T) -> Unit): T
inline fun <T, R> T.let(block: (T) -> R): R
inline fun <T> T.takeIf(predicate: (T) -> Boolean): T?
inline fun <T> T.takeUnless(predicate: (T) -> Boolean): T?
inline fun repeat(times: Int, action: (Int) -> Unit)
```

# kotlin inline functions

```kotlin
inline fun TODO(): Nothing
inline fun TODO(reason: String): Nothing
inline fun <R> run(block: () -> R): R
inline fun <T, R> T.run(block: T.() -> R): R
inline fun <T, R> with(receiver: T, block: T.() -> R): R
inline fun <T> T.apply(block: T.() -> Unit): T
inline fun <T> T.also(block: (T) -> Unit): T
inline fun <T, R> T.let(block: (T) -> R): R
inline fun <T> T.takeIf(predicate: (T) -> Boolean): T?
inline fun <T> T.takeUnless(predicate: (T) -> Boolean): T?
inline fun repeat(times: Int, action: (Int) -> Unit)
```

# kotlin inline functions

```
inline fun TODO(): Nothing
inline fun TODO(reason: String): Nothing
inline fun <R> run(block: () -> R): R
inline fun <T, R> T.run(block: T.() -> R): R
inline fun <T, R> with(receiver: T, block: T.() -> R): R
inline fun <T> T.apply(block: T.() -> Unit): T
inline fun <T> T.also(block: (T) -> Unit): T
inline fun <T, R> T.let(block: (T) -> R): R
inline fun <T> T.takeIf(predicate: (T) -> Boolean): T?
inline fun <T> T.takeUnless(predicate: (T) -> Boolean): T?
inline fun repeat(times: Int, action: (Int) -> Unit)
```

## TODO

```kotlin
inline fun TODO(): Nothing
inline fun TODO(reason: String): Nothing
```

# TODO

```
inline fun TODO(): Nothing
inline fun TODO(reason: String): Nothing
```

```
fun mock(){
    TODO("...")
}
mock()
```

## run

```kotlin
inline fun <R> run(block: () -> R): R
inline fun <T, R> T.run(block: T.() -> R): R
```

# run

```
inline fun <R> run(block: () -> R): R
inline fun <T, R> T.run(block: T.() -> R): R
```

```
val run0 = run{
    val a = 3
    val b = 5
    3 + 5
}
val run1 = 15.run{
    this + 10
}
```

# with

```kotlin
inline fun <T, R> with(receiver: T, block: T.() -> R): R
```

# with

```
inline fun <T, R> with(receiver: T, block: T.() -> R): R
```

```
val list1 = mutableListOf<String>()
val with1 = with(list1){
    list1.addAll("1,2,3,4,5,6,7".split(","))
    list1[0]
}
```

# apply

```kotlin
inline fun <T> T.apply(block: T.() -> Unit): T
```

# apply

```
inline fun <T> T.apply(block: T.() -> Unit): T
```

```
val apply1 = mutableListOf(1,2,3).apply {
    forEachIndexed{idx, v->
        this[idx] = v * 2
    }
}
```

## also

```
inline fun <T> T.also(block: (T) -> Unit): T
```

# also

```kotlin
inline fun <T> T.also(block: (T) -> Unit): T
```

```kotlin
val also1 = mutableListOf(1,2,3).also{
    it.forEachIndexed{idx, v->
        it[idx] = v * 2
    }
}
```

## let

```kotlin
inline fun <T, R> T.let(block: (T) -> R): R
```

# let

```
inline fun <T, R> T.let(block: (T) -> R): R
```

```
val v1:Int? = null
if(v1 != null) println("${double(v1)}")
```

# let

```kotlin
inline fun <T, R> T.let(block: (T) -> R): R
```

```kotlin
val v1:Int? = null
v1?.let{
    println("${double(v1)}")
}
```

## let

```
inline fun <T, R> T.let(block: (T) -> R): R
```

```
val v1:Int? = null
v1?.let{
    println("${double(v1)}")
}
val v2 = v1?.let{double(it)} ?: 0
```

# let

```
inline fun <T, R> T.let(block: (T) -> R): R
```

```
val v1:Int? = null
v1?.let{
    println("${double(v1)}")
}
val v2 = v1?.let{double(it)} ?: 0
```

Elvis operator

## takeIf & takeUnless

```kotlin
inline fun <T> T.takeIf(predicate: (T) -> Boolean): T?
inline fun <T> T.takeUnless(predicate: (T) -> Boolean): T?
```

# takeIf & takeUnless

```
inline fun <T> T.takeIf(predicate: (T) -> Boolean): T?
inline fun <T> T.takeUnless(predicate: (T) -> Boolean): T?
```

```
val takeList = mutableListOf(1,2,3)
val takeIf0 = if(takeList.size > 2) takeList else null
```

# takeIf & takeUnless

```kotlin
inline fun <T> T.takeIf(predicate: (T) -> Boolean): T?
inline fun <T> T.takeUnless(predicate: (T) -> Boolean): T?
```

```kotlin
val takeList = mutableListOf(1,2,3)
val takeIf0 = if(takeList.size > 2) takeList else null
val takeIf1 = takeList.takeIf {it.size > 2}
```

# takeIf & takeUnless

```
inline fun <T> T.takeIf(predicate: (T) -> Boolean): T?
inline fun <T> T.takeUnless(predicate: (T) -> Boolean): T?
```

```
val takeList = mutableListOf(1,2,3)
val takeIf0 = if(takeList.size > 2) takeList else null
val takeIf1 = takeList.takeIf {it.size > 2}

val takeUnless0  = if(takeList.size > 2) null else takeList
```

# takeIf & takeUnless

```
inline fun <T> T.takeIf(predicate: (T) -> Boolean): T?
inline fun <T> T.takeUnless(predicate: (T) -> Boolean): T?
```

```
val takeList = mutableListOf(1,2,3)
val takeIf0 = if(takeList.size > 2) takeList else null
val takeIf1 = takeList.takeIf {it.size > 2}

val takeUnless0  = if(takeList.size > 2) null else takeList
val takeUnless1 = takeList.takeUnless {it.size > 2}
```

## repeat

```
inline fun repeat(times: Int, action: (Int) -> Unit)
```

## repeat

```kotlin
inline fun repeat(times: Int, action: (Int) -> Unit)
```

```kotlin
var i = 0
while(i < 10){
    println(i)
    i++
}
```

## repeat

```
inline fun repeat(times: Int, action: (Int) -> Unit)
```

```
var i = 0
while(i < 10){
    println(i)
    i++
}
repeat(10){
    println(it)
}
```

## Request Builder

## Request Builder

```
val request = RequestBuilder("http://apiServer")
        .method(Method.POST)
        .form("name", "hika")
        .form("email", "hika@bsidesoft.com")
        .timeout(5000)
        .ok{}
        .fail{}
        .build()
```

# Request Builder

```kotlin
val request = RequestBuilder("http://apiServer")
        .method(Method.POST)
        .form("name", "hika")
        .form("email", "hika@bsidesoft.com")
        .timeout(5000)
        .ok{}
        .fail{}
        .build()

enum class Method{POST, GET}
```

# Request Builder

```kotlin
val request = RequestBuilder("http://apiServer")
        .method(Method.POST)
        .form("name", "hika")
        .form("email", "hika@bsidesoft.com")
        .timeout(5000)
        .ok{}
        .fail{}
        .build()
```

```kotlin
enum class Method{POST, GET}
```

```kotlin
class Request(
 val url:String,
 val method:Method,
 val form:MutableMap<String, String>?,
 val timeout:Int,
 val ok:((String) -> Unit)?,
 val fail:((String) -> Unit)?
)
```

# Request Builder

```
val request = RequestBuilder("http://apiServer")
        .method(Method.POST)
        .form("name", "hika")
        .form("email", "hika@bsidesoft.com")
        .timeout(5000)
        .ok{}
        .fail{}
        .build()
```

```
enum class Method{POST, GET}
```

```
class Request(
 val url:String,
 val method:Method,
 val form:MutableMap<String, String>?,
 val timeout:Int,
 val ok:((String) -> Unit)?,
 val fail:((String) -> Unit)?
)
```

```
class RequestBuilder(private val url:String){
  private var method: Method = Method.GET
  private val form = mutableMapOf<String, String>()
  private var timeout = 0
  private var ok:((String)->Unit)? = null
  private var fail:((String)->Unit)? = null
  fun method(method: Method):RequestBuilder{
    this.method = method
    return this
  }
  fun form(key:String, value:String):RequestBuilder{
    this.form[key] = value
    return this
  }
  fun timeout(ms:Int):RequestBuilder{
    this.timeout = ms
    return this
  }
  fun ok(block:(String)->Unit):RequestBuilder{
    this.ok = block
    return this
  }
  fun fail(block:(String)->Unit):RequestBuilder{
    this.fail = block
    return this
  }
  fun build() = Request(url, method, if(form.isEmpty()) null else form, timeout, ok, fail)
}
```

# takeIf

```kotlin
val request = RequestBuilder("http://apiServer")
        .method(Method.POST)
        .form("name", "hika")
        .form("email", "hika@bsidesoft.com")
        .timeout(5000)
        .ok{}
```

```kotlin
fun build() = Request(url, method, if(form.isEmpty()) null else form, timeout, ok, fail)
```

```kotlin
enum class Method{POST, GET}

class Request(
 val url:String,
 val method:Method,
 val form:MutableMap<String, String>?,
 val timeout:Int,
 val ok:((String) -> Unit)?,
 val fail:((String) -> Unit)?
)
```

```kotlin
class RequestBuilder(private val url:String){
  private var method: Method = Method.GET
  private val form = mutableMapOf<String, String>()
  private var timeout = 0
  private var ok:((String)->Unit)? = null
  private var fail:((String)->Unit)? = null
  fun method(method: Method):RequestBuilder{
    return this
  }
  fun form(key:String, value:String):RequestBuilder{
    this.form[key] = value
    return this
  }
  fun timeout(ms:Int):RequestBuilder{
    this.timeout = ms
    return this
  }
  fun ok(block:(String)->Unit):RequestBuilder{
    this.ok = block
    return this
  }
  fun fail(block:(String)->Unit):RequestBuilder{
    this.fail = block
    return this
  }
  fun build() = Request(url, method, if(form.isEmpty()) null else form, timeout, ok, fail)
}
```

# takeIf

```kotlin
val request = RequestBuilder("http://apiServer")
        .method(Method.POST)
        .form("name", "hika")
        .form("email", "hika@bsidesoft.com")
        .timeout(5000)
        .ok{}
```

```kotlin
fun build() = Request(url, method, if(form.isEmpty()) null else form, timeout, ok, fail)
```

```kotlin
enum class Method{POST, GET}

class Request(
 val url:String,
 val method:Method,
 val form:MutableMap<String, String>?,
 val timeout:Int,
 val ok:((String) -> Unit)?,
 val fail:((String) -> Unit)?
)
```

```kotlin
class RequestBuilder(private val url:String){
  private var method: Method = Method.GET
  private val form = mutableMapOf<String, String>()
  private var timeout = 0
  private var ok:((String)->Unit)? = null
  private var fail:((String)->Unit)? = null
  }
  fun form(key:String, value:String):RequestBuilder{
    this.form[key] = value
    return this
  }
  fun timeout(ms:Int):RequestBuilder{
    this.timeout = ms
    return this
  }
  fun ok(block:(String)->Unit):RequestBuilder{
    this.ok = block
    return this
  }
  fun fail(block:(String)->Unit):RequestBuilder{
    this.fail = block
    return this
  }
  fun build() = Request(url, method, if(form.isEmpty()) null else form, timeout, ok, fail)
}
```

# takeIf

```kotlin
fun build() = Request(url, method, if(form.isEmpty()) null else form, timeout, ok, fail)
```

```kotlin
fun build() = Request(url, method, form.takeIf{it.isNotEmpty()}, timeout, ok, fail)
```

```kotlin
val request = RequestBuilder("http://apiServer")
        .method(Method.POST)
        .form("name", "hika")
        .form("email", "hika@bsidesoft.com")
        .timeout(5000)
        .ok{}

enum class Method{POST, GET}

cl
  val url:String,
  val method:Method,
  val form:MutableMap<String, String>?,
  val timeout:Int,
  val ok:((String) -> Unit)?,
  val fail:((String) -> Unit)?
)
```

```kotlin
class RequestBuilder(private val url:String){
  private var method: Method = Method.GET
  private val form = mutableMapOf<String, String>()
  private var timeout = 0
  private var ok:((String)->Unit)? = null
  private var fail:((String)->Unit)? = null
  fun method(method: Method):RequestBuilder{

    return this
  }
  fun form(key:String, value:String):RequestBuilder{

    return this
  }
  fun timeout(ms:Int):RequestBuilder{
    this.timeout = ms
    return this
  }
  fun ok(block:(String)->Unit):RequestBuilder{
    this.ok = block
    return this
  }
  fun fail(block:(String)->Unit):RequestBuilder{
    this.fail = block
    return this
  }
  fun build() = Request(url, method, if(form.isEmpty()) null else form, timeout, ok, fail)
}
```

# takeIf

```
val request = RequestBuilder("http://apiServer")
        .method(Method.POST)
        .form("name", "hika")
        .form("email", "hika@bsidesoft.com")
        .timeout(5000)
        .ok{}
        .fail{}
        .build()
```

```
enum class Method{POST, GET}

class Request(
 val url:String,
 val method:Method,
 val form:MutableMap<String, String>?,
 val timeout:Int,
 val ok:((String) -> Unit)?,
 val fail:((String) -> Unit)?
)
```

```
class RequestBuilder(private val url:String){
  private var method: Method = Method.GET
  private val form = mutableMapOf<String, String>()
  private var timeout = 0
  private var ok:((String)->Unit)? = null
  private var fail:((String)->Unit)? = null
  fun method(method: Method):RequestBuilder{
    this.method = method
    return this
  }
  fun form(key:String, value:String):RequestBuilder{
    this.form[key] = value
    return this
  }
  fun timeout(ms:Int):RequestBuilder{
    this.timeout = ms
    return this
  }
  fun ok(block:(String)->Unit):RequestBuilder{
    this.ok = block
    return this
  }
  fun fail(block:(String)->Unit):RequestBuilder{
    this.fail = block
    return this
  }
  fun build() = Request(url, method, form.takeIf{it.isNotEmpty()}, timeout, ok, fail)
}
```

# typealias

```
val request = RequestBuilder("http://apiServer")
        .method(Method.POST)
        .form("name", "hika")
        .form("email", "hika@bsidesoft.com")
        .timeout(5000)
        .ok{}
        .fail{}
        .build()
```

```
enum class Method{POST, GET}
```

```
class Request(
 val url:String,
 val method:Method,
 val form:MutableMap<String, String>?,
 val timeout:Int,
 val ok:((String) -> Unit)?,
 val fail:((String) -> Unit)?
)
```

```
class RequestBuilder(private val url:String){
  private var method: Method = Method.GET
  private val form = mutableMapOf<String, String>()
  private var timeout = 0
  private var ok:((String)->Unit)? = null
  private var fail:((String)->Unit)? = null
  fun method(method: Method):RequestBuilder{
    this.method = method
    return this
  }
  fun form(key:String, value:String):RequestBuilder{
    this.form[key] = value
    return this
  }
  fun timeout(ms:Int):RequestBuilder{
    this.timeout = ms
    return this
  }
  fun ok(block:(String)->Unit):RequestBuilder{
    this.ok = block
    return this
  }
  fun fail(block:(String)->Unit):RequestBuilder{
    this.fail = block
    return this
  }
  fun build() = Request(url, method, form.takeIf{it.isNotEmpty()}, timeout, ok, fail)
}
```

# typealias

```kotlin
val request = RequestBuilder("http://apiServer")
        .method(Method.POST)
        .form("name", "hika")
        .form("email", "hika@bsidesoft.com")
        .timeout(5000)
        .ok{}
        .fail{}
        .build()

enum class Method{POST, GET}

class Request(
 val url:String,
 val method:Method,
 val form:MutableMap<String, String>?,
 val timeout:Int,
 val ok:((String) -> Unit)?,
 val fail:((String) -> Unit)?
)
```

```kotlin
class RequestBuilder(private val url:String){
  private var method: Method = Method.GET
  private val form = mutableMapOf<String, String>()
  private var timeout = 0
  private var ok:((String)->Unit)? = null
  private var fail:((String)->Unit)? = null
  fun method(method: Method):RequestBuilder{
    this.method = method
    return this
  }
  fun form(key:String, value:String):RequestBuilder{
    this.form[key] = value
    return this
  }
  fun timeout(ms:Int):RequestBuilder{
    this.timeout = ms
    return this
  }
  fun ok(block:(String)->Unit):RequestBuilder{
    this.ok = block
    return this
  }
  fun fail(block:(String)->Unit):RequestBuilder{
    this.fail = block
    return this
  }
  fun build() = Request(url, method, form.takeIf{it.isNotEmpty()}, timeout, ok, fail)
}
```

# typealias

```kotlin
val request = RequestBuilder("http://apiServer")
        .method(Method.POST)
        .form("name", "hika")
        .form("email", "hika@bsidesoft.com")
        .timeout(5000)
        .ok{}
        .fail{}
        .build()
```

```kotlin
typealias listener = (String) -> Unit
```

```kotlin
enum class Method{POST, GET}

class Request(
 val url:String,
 val method:Method,
 val form:MutableMap<String, String>?,
 val timeout:Int,
 val ok:((String) -> Unit)?,
 val fail:((String) -> Unit)?
)
```

```kotlin
class RequestBuilder(private val url:String){
  private var method: Method = Method.GET
  private val form = mutableMapOf<String, String>()
  private var timeout = 0
  private var ok:((String)->Unit)? = null
  private var fail:((String)->Unit)? = null
  fun method(method: Method):RequestBuilder{
    this.method = method
    return this
  }
  fun form(key:String, value:String):RequestBuilder{
    this.form[key] = value
    return this
  }
  fun timeout(ms:Int):RequestBuilder{
    this.timeout = ms
    return this
  }
  fun ok(block:(String)->Unit):RequestBuilder{
    this.ok = block
    return this
  }
  fun fail(block:(String)->Unit):RequestBuilder{
    this.fail = block
    return this
  }
  fun build() = Request(url, method, form.takeIf{it.isNotEmpty()}, timeout, ok, fail)
}
```

# typealias

```kotlin
val request = RequestBuilder("http://apiServer")
        .method(Method.POST)
        .form("name", "hika")
        .form("email", "hika@bsidesoft.com")
        .timeout(5000)
        .ok{}
        .fail{}
        .build()
```

```kotlin
typealias listener = (String) -> Unit
```

```kotlin
enum class Method{POST, GET}
```

```kotlin
class Request(
 val url:String,
 val method:Method,
 val form:MutableMap<String, String>?,
 val timeout:Int,
 val ok:listener?,
 val fail:listener?
)
```

```kotlin
class RequestBuilder(private val url:String){
  private var method: Method = Method.GET
  private val form = mutableMapOf<String, String>()
  private var timeout = 0
  private var ok:listener? = null
  private var fail:listener? = null
  fun method(method: Method):RequestBuilder{
    this.method = method
    return this
  }
  fun form(key:String, value:String):RequestBuilder{
    this.form[key] = value
    return this
  }
  fun timeout(ms:Int):RequestBuilder{
    this.timeout = ms
    return this
  }
  fun ok(block:listener):RequestBuilder{
    this.ok = block
    return this
  }
  fun fail(block:listener):RequestBuilder{
    this.fail = block
    return this
  }
  fun build() = Request(url, method, form.takeIf{it.isNotEmpty()}, timeout, ok, fail)
}
```

# run

```kotlin
val request = RequestBuilder("http://apiServer")
        .method(Method.POST)
        .form("name", "hika")
        .form("email", "hika@bsidesoft.com")
        .timeout(5000)
        .ok{}
        .fail{}
        .build()

typealias listener = (String) -> Unit

enum class Method{POST, GET}

class Request(
 val url:String,
 val method:Method,
 val form:MutableMap<String, String>?,
 val timeout:Int,
 val ok:listener?,
 val fail:listener?
)
```

```kotlin
class RequestBuilder(private val url:String){
  private var method: Method = Method.GET
  private val form = mutableMapOf<String, String>()
  private var timeout = 0
  private var ok:listener? = null
  private var fail:listener? = null
  fun method(method: Method):RequestBuilder{
    this.method = method
    return this
  }
  fun form(key:String, value:String):RequestBuilder{
    this.form[key] = value
    return this
  }
  fun timeout(ms:Int):RequestBuilder{
    this.timeout = ms
    return this
  }
  fun ok(block:listener):RequestBuilder{
    this.ok = block
    return this
  }
  fun fail(block:listener):RequestBuilder{
    this.fail = block
    return this
  }
  fun build() = Request(url, method, form.takeIf{it.isNotEmpty()}, timeout, ok, fail)
}
```

# run

```kotlin
val request = RequestBuilder("http://apiServer")
        .method(Method.POST)
        .form("name", "hika")
        .form("email", "hika@bsidesoft.com")
        .timeout(5000)
        .ok{}
        .fail{}
        .build()
```

```kotlin
typealias listener = (String) -> Unit
```

```kotlin
enum class Method{POST, GET}
```

```kotlin
class Request(
 val url:String,
 val method:Method,
 val form:MutableMap<String, String>?,
 val timeout:Int,
 val ok:listener?,
 val fail:listener?
)
```

```kotlin
class RequestBuilder(private val url:String){
  private var method: Method = Method.GET
  private val form = mutableMapOf<String, String>()
  private var timeout = 0
  private var ok:listener? = null
  private var fail:listener? = null
  fun method(method: Method) = run{
    this.method = method
    this
  }
  fun form(key:String, value:String):RequestBuilder{
    this.form[key] = value
    return this
  }
  fun timeout(ms:Int):RequestBuilder{
    this.timeout = ms
    return this
  }
  fun ok(block:listener):RequestBuilder{
    this.ok = block
    return this
  }
  fun fail(block:listener):RequestBuilder{
    this.fail = block
    return this
  }
  fun build() = Request(url, method, form.takeIf{it.isNotEmpty()}, timeout, ok, fail)
}
```

# run

```kotlin
val request = RequestBuilder("http://apiServer")
        .method(Method.POST)
        .form("name", "hika")
        .form("email", "hika@bsidesoft.com")
        .timeout(5000)
        .ok{}
        .fail{}
        .build()
```

```kotlin
typealias listener = (String) -> Unit
```

```kotlin
enum class Method{POST, GET}
```

```kotlin
class Request(
 val url:String,
 val method:Method,
 val form:MutableMap<String, String>?,
 val timeout:Int,
 val ok:listener?,
 val fail:listener?
)
```

```kotlin
class RequestBuilder(private val url:String){
  private var method: Method = Method.GET
  private val form = mutableMapOf<String, String>()
  private var timeout = 0
  private var ok:listener? = null
  private var fail:listener? = null
  fun method(method: Method) = run{
    this.method = method
    this
  }
  fun form(key:String, value:String) = run{
    this.form[key] = value
    this
  }
  fun timeout(ms:Int) = run{
    this.timeout = ms
    this
  }
  fun ok(block:listener) = run{
    this.ok = block
    this
  }
  fun fail(block:listener) = run{
    this.fail = block
    this
  }
  fun build() = Request(url, method, form.takeIf{it.isNotEmpty()}, timeout, ok, fail)
}
```

# with

```
val request = RequestBuilder("http://apiServer")
          .method(Method.POST)
          .form("name", "hika")
          .form("email", "hika@bsidesoft.com")
          .timeout(5000)
          .ok{}
          .fail{}
          .build()
```

```
class Request(
 val url:String,
 val method:Method,
 val form:MutableMap<String, String>?,
 val timeout:Int,
 val ok:listener?,
 val fail:listener?
)
```

```
private val url:String){
 Method = Method.GET
mutableMapOf<String, String>()
 = 0
ener? = null
stener? = null
Method) = run{
hod


, value:String) = run{
value

  }
  fun timeout(ms:Int) = run{
    this.timeout = ms
    this
  }
  fun ok(block:listener) = run{
    this.ok = block
    this
  }
  fun fail(block:listener) = run{
    this.fail = block
    this
  }
  fun build() = Request(url, method, form.takeIf{it.isNotEmpty()}, timeout, ok, fail)
}
```

# with

```kotlin
val request = RequestBuilder("http://apiServer")
        .method(Method.POST)
        .form("name", "hika")
        .form("email", "hika@bsidesoft.com")
        .timeout(5000)
        .ok{}
        .fail{}
        .build()
```

```kotlin
private val url:String){
 Method = Method.GET
mutableMapOf<String, String>()
 = 0
ener? = null
stener? = null
Method) = run{
hod


, value:String) = run{
value
```

```kotlin
class Request(
 val url:String,
 val method:Method,
 val form:MutableMap<String, String>?,
 val timeout:Int,
 val ok:listener?,
 val fail:listener?
)
```

```kotlin
val request = with(RequestBuilder("http://apiServer")) {
    method(Method.POST)
    form("name", "hika")
    form("email", "hika@bsidesoft.com")
    timeout(5000)
    ok {}
    fail {}
    build()
}
```

# with

```kotlin
val request = with(RequestBuilder("http://apiServer")) {
    method(Method.POST)
    form("name", "hika")
    form("email", "hika@bsidesoft.com")
    timeout(5000)
    ok {}
    fail {}
    build()
}
```

```kotlin
typealias listener = (String) -> Unit
```

```kotlin
enum class Method{POST, GET}
```

```kotlin
class Request(
 val url:String,
 val method:Method,
 val form:MutableMap<String, String>?,
 val timeout:Int,
 val ok:listener?,
 val fail:listener?
)
```

```kotlin
class RequestBuilder(private val url:String){
  private var method: Method = Method.GET
  private val form = mutableMapOf<String, String>()
  private var timeout = 0
  private var ok:listener? = null
  private var fail:listener? = null
  fun method(method: Method) = run{
    this.method = method
    this
  }
  fun form(key:String, value:String) = run{
    this.form[key] = value
    this
  }
  fun timeout(ms:Int) = run{
    this.timeout = ms
    this
  }
  fun ok(block:listener) = run{
    this.ok = block
    this
  }
  fun fail(block:listener) = run{
    this.fail = block
    this
  }
  fun build() = Request(url, method, form.takeIf{it.isNotEmpty()}, timeout, ok, fail)
}
```

# with

```kotlin
val request = with(RequestBuilder("http://apiServer")) {
    method(Method.POST)
    form("name", "hika")
    form("email", "hika@bsidesoft.com")
    timeout(5000)
    ok {}
    fail {}
    build()
}
```

```kotlin
typealias listener = (String) -> Unit
```

```kotlin
enum class Method{POST, GET}
```

```kotlin
class Request(
 val url:String,
 val method:Method,
 val form:MutableMap<String, String>?,
 val timeout:Int,
 val ok:listener?,
 val fail:listener?
)
```

```kotlin
class RequestBuilder(private val url:String){
  private var method: Method = Method.GET
  private val form = mutableMapOf<String, String>()
  private var timeout = 0
  private var ok:listener? = null
  private var fail:listener? = null
  fun method(method: Method){this.method = method}
  fun form(key:String, value:String) = run{
    this.form[key] = value
    this
  }
  fun timeout(ms:Int) = run{
    this.timeout = ms
    this
  }
  fun ok(block:listener) = run{
    this.ok = block
    this
  }
  fun fail(block:listener) = run{
    this.fail = block
    this
  }
  fun build() = Request(url, method, form.takeIf{it.isNotEmpty()}, timeout, ok, fail)
}
```

# with

```kotlin
val request = with(RequestBuilder("http://apiServer")) {
    method(Method.POST)
    form("name", "hika")
    form("email", "hika@bsidesoft.com")
    timeout(5000)
    ok {}
    fail {}
    build()
}
```

```kotlin
typealias listener = (String) -> Unit
enum class Method{POST, GET}
class Request(
 val url:String,
 val method:Method,
 val form:MutableMap<String, String>?,
 val timeout:Int,
 val ok:listener?,
 val fail:listener?
)
```

```kotlin
class RequestBuilder(private val url:String){
    private var method: Method = Method.GET
    private val form = mutableMapOf<String, String>()
    private var timeout = 0
    private var ok:listener? = null
    private var fail:listener? = null
    fun method(method: Method){this.method = method}
    fun form(key:String, value:String){this.form[key] = value}
    fun timeout(ms:Int){this.timeout = ms}
    fun ok(block:listener){this.ok = block}
    fun fail(block:listener){this.fail = block}
    fun build() = Request(url, method, form.takeIf{it.isNotEmpty()}, timeout, ok, fail)
}
```

# extentions

```
val request = with(RequestBuilder("http://apiServer")) {
    method(Method.POST)
    form("name", "hika")
    form("email", "hika@bsidesoft.com")
    timeout(5000)
    ok {}
    fail {}
    build()
}
```

```
class RequestBuilder(private val url:String){
    private var method: Method = Method.GET
    private val form = mutableMapOf<String, String>()
    private var timeout = 0
    private var ok:listener? = null
    private var fail:listener? = null
    fun method(method: Method){this.method = method}
    fun form(key:String, value:String){this.form[key] = value}
    fun timeout(ms:Int){this.timeout = ms}
    fun ok(block:listener){this.ok = block}
    fun fail(block:listener){this.fail = block}
    fun build() = Request(url, method, form.takeIf{it.isNotEmpty()}, timeout, ok, fail)
}
```

```
typealias listener = (String) -> Unit

enum class Method{POST, GET}

class Request(
 val url:String,
 val method:Method,
 val form:MutableMap<String, String>?,
 val timeout:Int,
 val ok:listener?,
 val fail:listener?
)
```

# extentions

```
val request = with(RequestBuilder("http://apiServer")) {
```

```
val request = with(RequestBuilder("http://apiServer")) {
    method(Method.POST)
    form("name", "hika")
    form("email", "hika@bsidesoft.com")
    timeout(5000)
    ok {}
    fail {}
    build()
}
```

```
rl:String){
lethod.GET
Of<String, String>()

l
null
s.method = method}
ing){this.form[key] = value}
out = ms}
 = block}
 fail = block}
hod, form.takeIf{it.isNotEmpty()}, timeout, ok, fail)
```

```
 val url:String,
 val method:Method,
 val form:MutableMap<String, String>?,
 val timeout:Int,
 val ok:listener?,
 val fail:listener?
)
```

# extentions

```
val request = with(RequestBuilder("http://apiServer")) {
```

```
val request = with(RequestBuilder("http://apiServer")) {
    method(Method.POST)
    form("name", "hika")
    form("email", "hika@bsidesoft.com")
    timeout(5000)
    ok {}
    fail {}
    build()
}
```

```
rl:String){
lethod.GET
Of<String, String>()

l
ull
s.method = method}
ing){this.form[key] = value}
out = ms}
 = block}
 fail = block}
hod, form.takeIf{it.isNotEmpty()}, timeout, ok, fail)
```

```
 val url:String,
 val method:Method,
 val form:MutableMap<String, String>?,
 val timeout:Int,
 val ok:listener?,
 val fail:listener?
)
```

```
val request = RequestBuilder("http://apiServer"){
  method = Method.POST
  form["name"] = "hika"
  form["email"] = "hika@bsidesoft.com"
  timeout = 5000
  ok = {}
  fail = {}
}
```

# extentions

```
val request = RequestBuilder("http://apiServer"){
  method = Method.POST
  form["name"] = "hika"
  form["email"] = "hika@bsidesoft.com"
  timeout = 5000
  ok = {}
  fail = {}
}
```

```
typealias listener = (String) -> Unit
```

```
enum class Method{POST, GET}
```

```
class Request(
 val url:String,
 val method:Method,
 val form:MutableMap<String, String>?,
 val timeout:Int,
 val ok:listener?,
 val fail:listener?
)
```

```
fun RequestBuilder(url:String, block:RequestBuilder.()->Unit)
  = RequestBuilder(url).apply(block).build()
```

```
class RequestBuilder(private val url:String){
  private var method: Method = Method.GET
  private val form = mutableMapOf<String, String>()
  private var timeout = 0
  private var ok:listener? = null
  private var fail:listener? = null
  fun method(method: Method){this.method = method}
  fun form(key:String, value:String){this.form[key] = value}
  fun timeout(ms:Int){this.timeout = ms}
  fun ok(block:listener){this.ok = block}
  fun fail(block:listener){this.fail = block}
  fun build() = Request(url, method, form.takeIf{it.isNotEmpty()}, timeout, ok, fail)
}
```

# extentions

```kotlin
val request = RequestBuilder("http://apiServer"){
  method = Method.POST
  form["name"] = "hika"
  form["email"] = "hika@bsidesoft.com"
  timeout = 5000
  ok = {}
  fail = {}
}
```

```kotlin
typealias listener = (String) -> Unit

enum class Method{POST, GET}

class Request(
 val url:String,
 val method:Method,
 val form:MutableMap<String, String>?,
 val timeout:Int,
 val ok:listener?,
 val fail:listener?
)
```

```kotlin
fun RequestBuilder(url:String, block:RequestBuilder.()->Unit)
  = RequestBuilder(url).apply(block).run{
     Request(url, method, form.takeIf{it.isNotEmpty()}, timeout, ok, fail)
  }

class RequestBuilder(private val url:String){
  var method: Method = Method.GET
  private val form = mutableMapOf<String, String>()
  fun form(key:String, value:String){this.form[key] = value}
  var timeout = 0
  var ok:listener? = null
  var fail:listener? = null
}
```

## HTML parser

## HTML PARSER

A = <TAG>BODY</TAG>

# HTML PARSER

```
A = <TAG>BODY</TAG>
B = <TAG/>
```

# HTML PARSER

A = <TAG>BODY</TAG>

B = <TAG/>

C = TEXT

# HTML PARSER

A = <TAG>BODY</TAG>

B = <TAG/>

C = TEXT

BODY = (A | B | C)N

## HTML PARSER

A = <TAG>BODY</TAG>

B = <TAG/>

C = TEXT

BODY = (A | B | C)N

```
<div>a<a>b</a>c<img/>d</div>
```

# HTML PARSER

A = <TAG>BODY</TAG>

B = <TAG/>

C = TEXT

BODY = (A | B | C)N

```
<div>a<a>b</a>c<img/>d</div>
<div>
  a
  <a>b</a>
  c
  <img/>
  d
</div>
```

# HTML PARSER

A = <TAG>BODY</TAG>

B = <TAG/>

C = TEXT

BODY = (A | B | C)N

```
<div>a<a>b</a>c<img/>d</div>
<div>
  a
  <a>b</a>
  c
  <img/>
  d
</div>
```

A

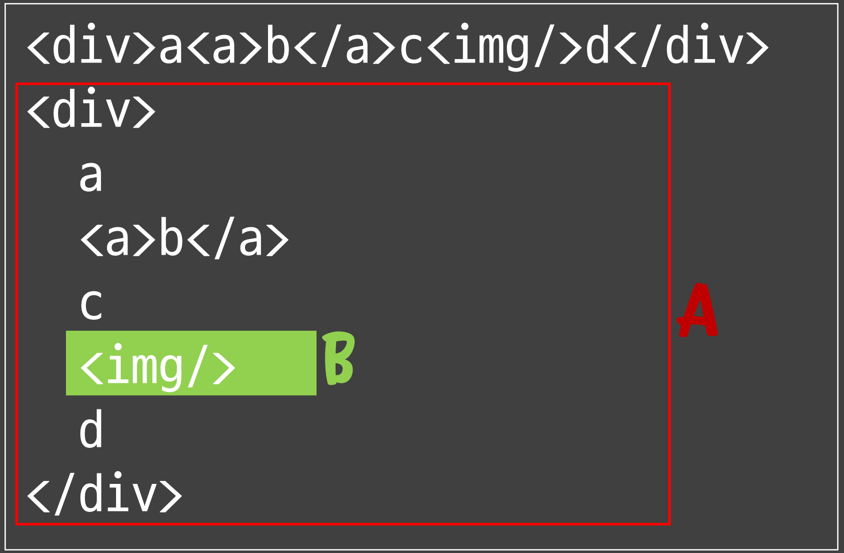
# HTML PARSER

A = <TAG>BODY</TAG>

B = <TAG/>

C = TEXT

BODY = (A | B | C)N



```
<div>a<a>b</a>c<img/>d</div>
<div>
  a
  <a>b</a>
  c
  <img/>
  d
</div>
```

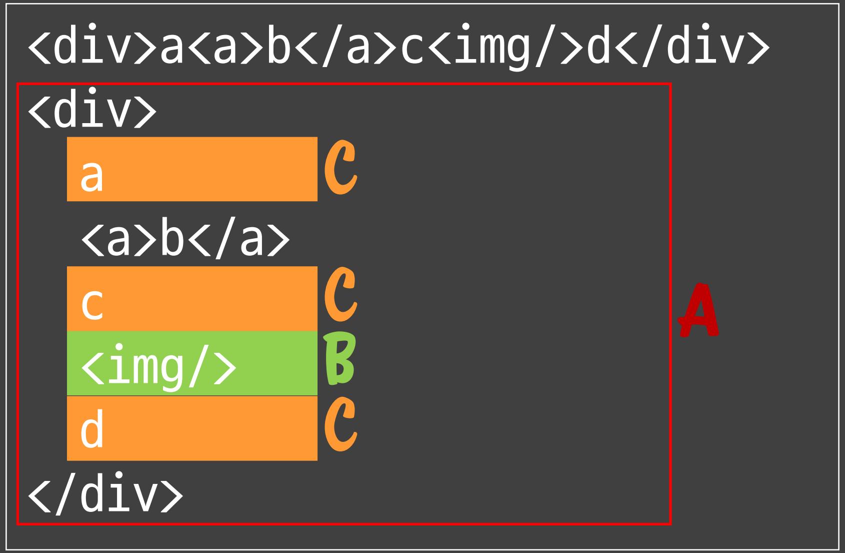
# HTML PARSER

A = <TAG>BODY</TAG>

B = <TAG/>

C = TEXT

BODY = (A | B | C)N



```
<div>a<a>b</a>c<img/>d</div>
<div>
  a
  <a>b</a>
  c
  <img/>
  d
</div>
```

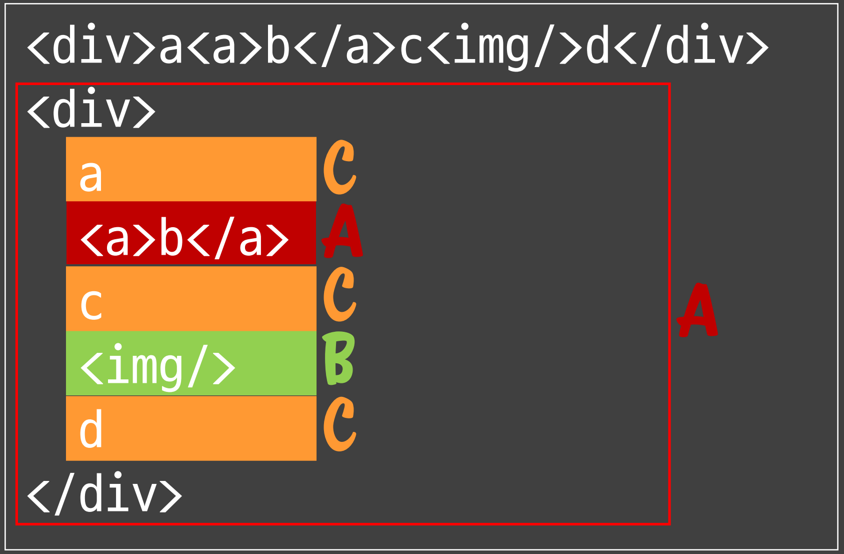
# HTML PARSER

A = <TAG>BODY</TAG>

B = <TAG/>

C = TEXT

BODY = (A | B | C)N



```
<div>a<a>b</a>c<img/>d</div>
```

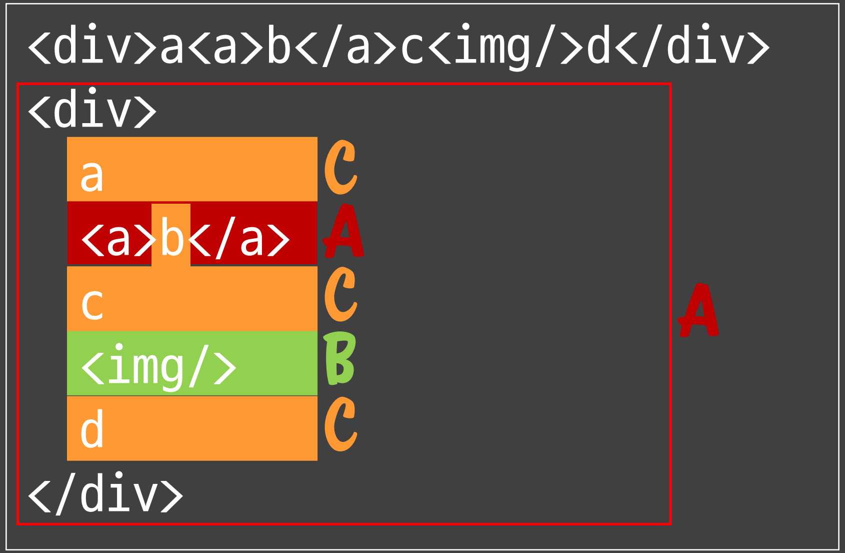
# HTML PARSER

A = <TAG>BODY</TAG>

B = <TAG/>

C = TEXT

BODY = (A | B | C)N



```
<div>a<a>b</a>c<img/>d</div>
```

## Data Structure

```kotlin
abstract class Node(val parent: Element?)
class Element(val tagName:String, parent:Element?):Node(parent){
    val attributes = mutableMapOf<String, String>()
    val children = mutableListOf<Node>()
}
class TextNode(val text:String, parent:Element?):Node(parent)
```

# Entry

```
abstract class Node(val parent: Element?)
class Element(val tagName:String, parent:Element?):Node(parent){
    val attributes = mutableMapOf<String, String>()
    val children = mutableListOf<Node>()
}
class TextNode(val text:String, parent:Element?):Node(parent)
```

```
fun parseHTML(v:String) = parse(Element("root", null), v)
```

# parse

A = <TAG>BODY</TAG>

B = <TAG/>

C = TEXT

BODY = (A | B | C)N

```
<div>
  a
  <a>b</a>
  c
  <img/>
  d
</div>
```

```
fun parseHTML(v:String) = parse(Element("root", null), v)

fun parse(parent:Element, v:String):Element{
  if(v[0] != '<'){
   C
  }else{
    val next = v.indexOf('>')
    if(v[1] == '/'){
      A의 닫기 경우
    }else{
      val isClose = v[next - 1] == '/'
      isClose면 B 아니면 A
    }
}
```

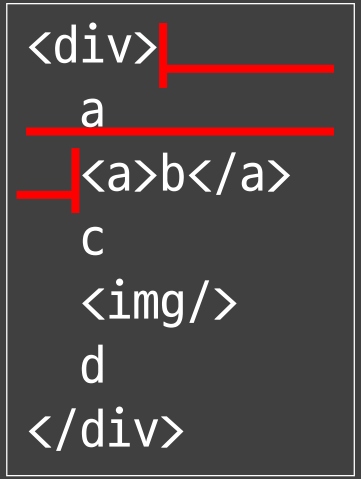
# parse - c:text

A = <TAG>BODY</TAG>

B = <TAG/>

C = TEXT

BODY = (A | B | C)N


```
<div>
  a
  <a>b</a>
  c
  <img/>
  d
</div>
```

```
fun parse(parent:Element, v:String):Element{
  if(v[0] != '<'){
    return if(v.isEmpty()) parent
    else{
      val next = v.indexOf('<')
      parent.children += TextNode(v.substring(0, if(next == -1) v.length else next), parent)
      if(next == -1) parent else parse(parent, v.substring(next))
    }
  }else{
    val next = v.indexOf('>')
    if(v[1] == '/'){
      A의 닫기 경우
    }else{
      val isClose = v[next - 1] == '/'
      isClose면 B 아니면 A
    }
}
```

# parse - c:text

A = <TAG>BODY</TAG>

B = <TAG/>

C = TEXT

BODY = (A | B | C)N

```
<div>
  a
  <a>b</a>
  c
  <img/>
  d
</div>
```

```
fun parse(parent:Element, v:String):Element{
  if(v[0] != '<'){
    return if(v.isEmpty()) parent
    else{
      val next = v.indexOf('<')
      parent.children += TextNode(v.substring(0, if(next == -1) v.length else next), parent)
      if(next == -1) parent else parse(parent, v.substring(next))
    }
  }else{
    val next = v.indexOf('>')
    if(v[1] == '/'){
      A의 닫기 경우
    }else{
      val isClose = v[next - 1] == '/'
      isClose면 B 아니면 A
    }
}
```

```
<p>test</p>
plain text
```

# parse - c:text

**A = <TAG>BODY</TAG>**

**B = <TAG/>**

**C = TEXT**

**BODY = (A | B | C)N**

```
<div>
  a
  <a>b</a>
  c
  <img/>
  d
</div>
```

```
fun parse(parent:Element, v:String):Element{
  if(v[0] != '<'){
    return if(v.isEmpty()) parent
    else{
      val next = v.indexOf('<')
      parent.children += TextNode(v.substring(0, if(next == -1) v.length else next), parent)
      if(next == -1) parent else parse(parent, v.substring(next))
    }
  }else{
    val next = v.indexOf('>')
    if(v[1] == '/'){
      A의 닫기 경우
    }else{
      val isClose = v[next - 1] == '/'
      isClose면 B 아니면 A
    }
}
```

# parse – closing tag

A = <TAG>BODY</TAG>

B = <TAG/>

C = TEXT

BODY = (A | B | C)N

```
<div>
  a
  <a>b</a>
  c
  <img/>
  d
</div>
```

```
fun parse(parent:Element, v:String):Element{
  if(v[0] != '<'){
    return if(v.isEmpty()) parent
    else{
      val next = v.indexOf('<')
      parent.children += TextNode(v.substring(0, if(next == -1) v.length else next), parent)
      if(next == -1) parent else parse(parent, v.substring(next))
    }
  }else{
    val next = v.indexOf('>')
    if(v[1] == '/'){
      return if(parent.parent == null) parent
      else parse(parent.parent, v.substring(next + 1))
    }else{
      val isClose = v[next - 1] == '/'
      isClose면 B 아니면 A
    }
}
```

# parse – closing tag

A = <TAG>BODY</TAG>

B = <TAG/>

C = TEXT

BODY = (A | B | C)N

```
<div>
  a
  <a>b</a>
  c
  <img/>
  d
</div>
```

```
fun parse(parent:Element, v:String):Element{
  if(v[0] != '<'){
    return if(v.isEmpty()) parent
    else{
      val next = v.indexOf('<')
      parent.children += TextNode(v.substring(0, if(next == -1) v.length else next), parent)
      if(next == -1) parent else parse(parent, v.substring(next))
    }
  }else{
    val next = v.indexOf('>')
    if(v[1] == '/'){
      return if(parent.parent == null) parent
      else parse(parent.parent, v.substring(next + 1))
    }else{
      val isClose = v[next - 1] == '/'
      isClose면 B 아니면 A
    }
}
```

# parse – closing tag

A = <TAG>BODY</TAG>

B = <TAG/>

C = TEXT

BODY = (A | B | C)N

```
<div>
  a
  <a>b</a>
  c
  <img/>
  d
</div>
```

```
fun parse(parent:Element, v:String):Element{
  if(v[0] != '<'){
    return if(v.isEmpty()) parent
    else{
      val next = v.indexOf('<')
      parent.children += TextNode(v.substring(0, if(next == -1) v.length else next), parent)
      if(next == -1) parent else parse(parent, v.substring(next))
    }
  }else{
    val next = v.indexOf('>')
    if(v[1] == '/'){
      return if(parent.parent == null) parent
      else parse(parent.parent, v.substring(next + 1))
    }else{
      val isClose = v[next - 1] == '/'
      isClose면 B 아니면 A
    }
}
```

## attibute

```
<div a="3" b="abc" diable>
```

```
val rex = """...""".toRegex()
```

## attibute

```
<div a="3" b="abc" diable>
```

```scala
val rex = """<([a-zA-Z]+)""".toRegex()
```

# attibute

```
<div a="3" b="abc" diable>
```

```
val rex = """<([a-zA-Z]+)""".toRegex()
```

```
val rex = """\s+[a-zA-Z-]+\s*=\s*"[^"]*"""".toRegex()
```

# attibute

```
<div a="3" b="abc" diable>
```

```scala
val rex = """<([a-zA-Z]+)""".toRegex()
```

```scala
val rex = """\s+[a-zA-Z-]+\s*=\s*"[^"]*"""".toRegex()
```

```scala
val rex = """\s+[a-zA-Z-]+""".toRegex()
```

# attibute

```
<div a="3" b="abc" diable>
```

```
val rex = """<([a-zA-Z]+)""".toRegex()
```

```
val rex = """\s+[a-zA-Z-]+\s*=\s*"[^"]*"""".toRegex()
```

```
val rex = """\s+[a-zA-Z-]+""".toRegex()
```

```
val rex = """\s+[a-zA-Z-]+(?:\s*=\s*"[^"]*")?""".toRegex()
```

# attibute

```
<div a="3" b="abc" diable>
```

```
val rex = """<([a-zA-Z]+)((?:\s+[a-zA-Z-]+(?:\s*=\s*"[^"]*")?)*)""".toRegex()
```

```
val rex = """\s+[a-zA-Z-]+\s*=\s*"[^"]*"""".toRegex()
```

```
val rex = """\s+[a-zA-Z-]+""".toRegex()
```

```
val rex = """\s+[a-zA-Z-]+(?:\s*=\s*"[^"]*")?""".toRegex()
```

# attibute

```
<div a="3" b="abc" diable>
```

```
val rex = """<([a-zA-Z]+)((?:\s+[a-zA-Z-]+(?:\s*=\s*"[^"]*")?)*)""".toRegex()
```

```
<div a="3" b="abc">
<div a="3" b="abc"     >
<div a="3" b="abc"/>
<div a="3" b="abc"    />
```

# attibute

```
<div a="3" b="abc" diable>
```

```
val rex = """<([a-zA-Z]+)((?:\s+[a-zA-Z-]+(?:\s*=\s*"[^"]*")?)*)\s*/?""".toRegex()
```

```
<div a="3" b="abc">
<div a="3" b="abc"    >
<div a="3" b="abc"/>
<div a="3" b="abc"   />
```

# parse – a, b open

A = <TAG>BODY</TAG>

B = <TAG/>

C = TEXT

BODY = (A | B | C)N

```
<div>
  a
  <a>b</a>
  c
  <img/>
  d
</div>
```

```
val rex = """<([a-zA-Z]+)((?:\s+[a-zA-Z-]+(?:\s*=\s*"[^"]*")?)*)\s*/?""".toRegex()
fun parse(parent:Element, v:String):Element{
  if(v[0] != '<'){
    return ...
  }else{
    val next = v.indexOf('>')
    if(v[1] == '/'){
      return ...
    }else{
      val isClose = v[next - 1] == '/'
      val matches = rex.matchEntire(v.substring(0, next))?.groupValues!!
      val el = Element(matches[1], parent)
      if(matches[2].isNotBlank()) matches[2].trim().split(' ').forEach {
        if(it.contains('=')) {
          val kv = it.split('=').map { it.trim() }
          el.attributes[kv[0]] = kv[1].replace("\"", "")
        }else el.attributes[it] = "true"
      }
      parent.children += el
      return parse(if(isClose) parent else el, v.substring(next + 1))
    }
}
```

# parse – a, b open

A = <TAG>BODY</TAG>

B = <TAG/>

C = TEXT

BODY = (A | B | C)N

```
<div>
  a
  <a>b</a>
  c
  <img/>
  d
</div>
```

```kotlin
val rex = """<([a-zA-Z]+)((?:\s+[a-zA-Z-]+(?:\s*=\s*"[^"]*")?)*)\s*/?""".toRegex()
fun parse(parent:Element, v:String):Element{
  if(v[0] != '<'){
    return ...
  }else{
    val next = v.indexOf('>')
    if(v[1] == '/'){
      return ...
    }else{
      val isClose = v[next - 1] == '/'
      val matches = rex.matchEntire(v.substring(0, next))?.groupValues!!
      val el = Element(matches[1], parent)
      if(matches[2].isNotBlank()) matches[2].trim().split(' ').forEach {
        if(it.contains('=')) {
          val kv = it.split('=').map { it.trim() }
          el.attributes[kv[0]] = kv[1].replace("\"", "")
        }else el.attributes[it] = "true"
      }
      parent.children += el
      return parse(if(isClose) parent else el, v.substring(next + 1))
    }
}
```

not-null assertion operator
throw an NPE

# parse – a, b open

A = <TAG>BODY</TAG>

B = <TAG/>

C = TEXT

BODY = (A | B | C)N

```
<div>
  a
  <a>b</a>
  c
  <img/>
  d
</div>
```

```kotlin
val rex = """<([a-zA-Z]+)((?:\s+[a-zA-Z-]+(?:\s*=\s*"[^"]*")?)*)\s*/?""".toRegex()
fun parse(parent:Element, v:String):Element{
  if(v[0] != '<'){
    return ...
  }else{
    val next = v.indexOf('>')
    if(v[1] == '/'){
      return ...
    }else{
      val isClose = v[next - 1] == '/'
      val matches = rex.matchEntire(v.substring(0, next))?.groupValues!!
      val el = Element(matches[1], parent)
      if(matches[2].isNotBlank()) matches[2].trim().split(' ').forEach {
        if(it.contains('=')) {
          val kv = it.split('=').map { it.trim() }
          el.attributes[kv[0]] = kv[1].replace("\"", "")
        }else el.attributes[it] = "true"
      }
      parent.children += el
      return parse(if(isClose) parent else el, v.substring(next + 1))
    }
}
```

# parse - a, b open

A = <TAG>BODY</TAG>

B = <TAG/>

C = TEXT

BODY = (A | B | C)N

```
<div>
  a
  <a>b</a>
  c
  <img/>
  d
</div>
```

```
val rex = """<([a-zA-Z]+)((?:\s+[a-zA-Z-]+(?:\s*=\s*"[^"]*")?)*)\s*/?""".toRegex()
fun parse(parent:Element, v:String):Element{
  if(v[0] != '<'){
    return ...
  }else{
    val next = v.indexOf('>')
    if(v[1] == '/'){
      return ...
    }else{
      val isClose = v[next - 1] == '/'
      val matches = rex.matchEntire(v.substring(0, next))?.groupValues!!
      val el = Element(matches[1], parent)
      if(matches[2].isNotBlank()) matches[2].trim().split(' ').forEach {
        if(it.contains('=')) {
          val kv = it.split('=').map { it.trim() }
          el.attributes[kv[0]] = kv[1].replace("\"", "")
        }else el.attributes[it] = "true"
      }
      parent.children += el
      return parse(if(isClose) parent else el, v.substring(next + 1))
    }
}
```

# parse – a, b open

A = <TAG>BODY</TAG>

B = <TAG/>

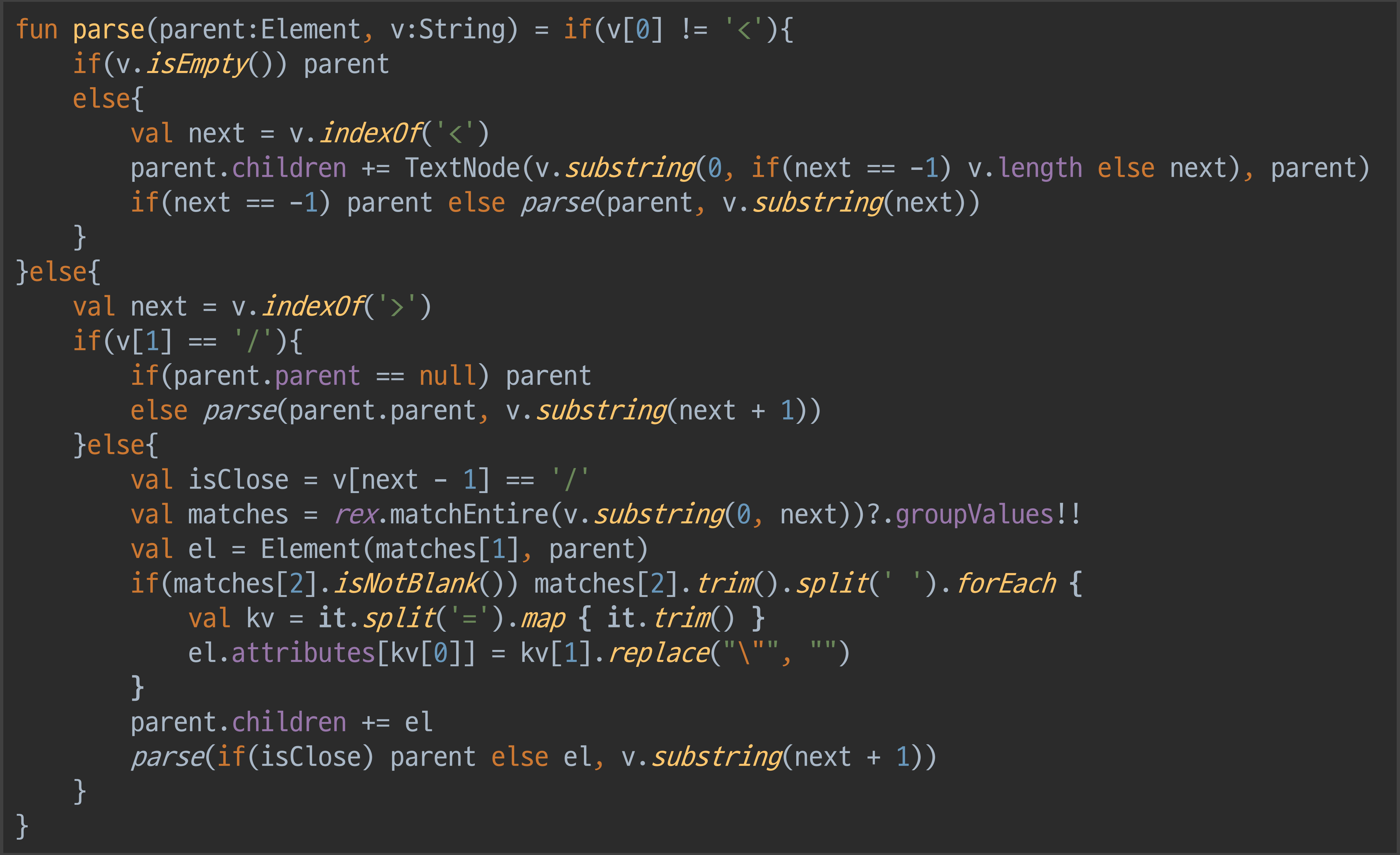
C = TEXT

BODY = (A | B | C)N

```
<div>
  a
  <a>b</a>
  c
  <img/>
  d
</div>
```

```kotlin
val rex = """<([a-zA-Z]+)((?:\s+[a-zA-Z-]+(?:\s*=\s*"[^"]*")?)*)\s*/?""".toRegex()
fun parse(parent:Element, v:String):Element{
  if(v[0] != '<'){
    return ...
  }else{
    val next = v.indexOf('>')
    if(v[1] == '/'){
      return ...
    }else{
      val isClose = v[next - 1] == '/'
      val matches = rex.matchEntire(v.substring(0, next))?.groupValues!!
      val el = Element(matches[1], parent)
      if(matches[2].isNotBlank()) matches[2].trim().split(' ').forEach {
        if(it.contains('=')) {
          val kv = it.split('=').map { it.trim() }
          el.attributes[kv[0]] = kv[1].replace("\"", "")
        }else el.attributes[it] = "true"
      }
      parent.children += el
      return parse(if(isClose) parent else el, v.substring(next + 1))
    }
}
```

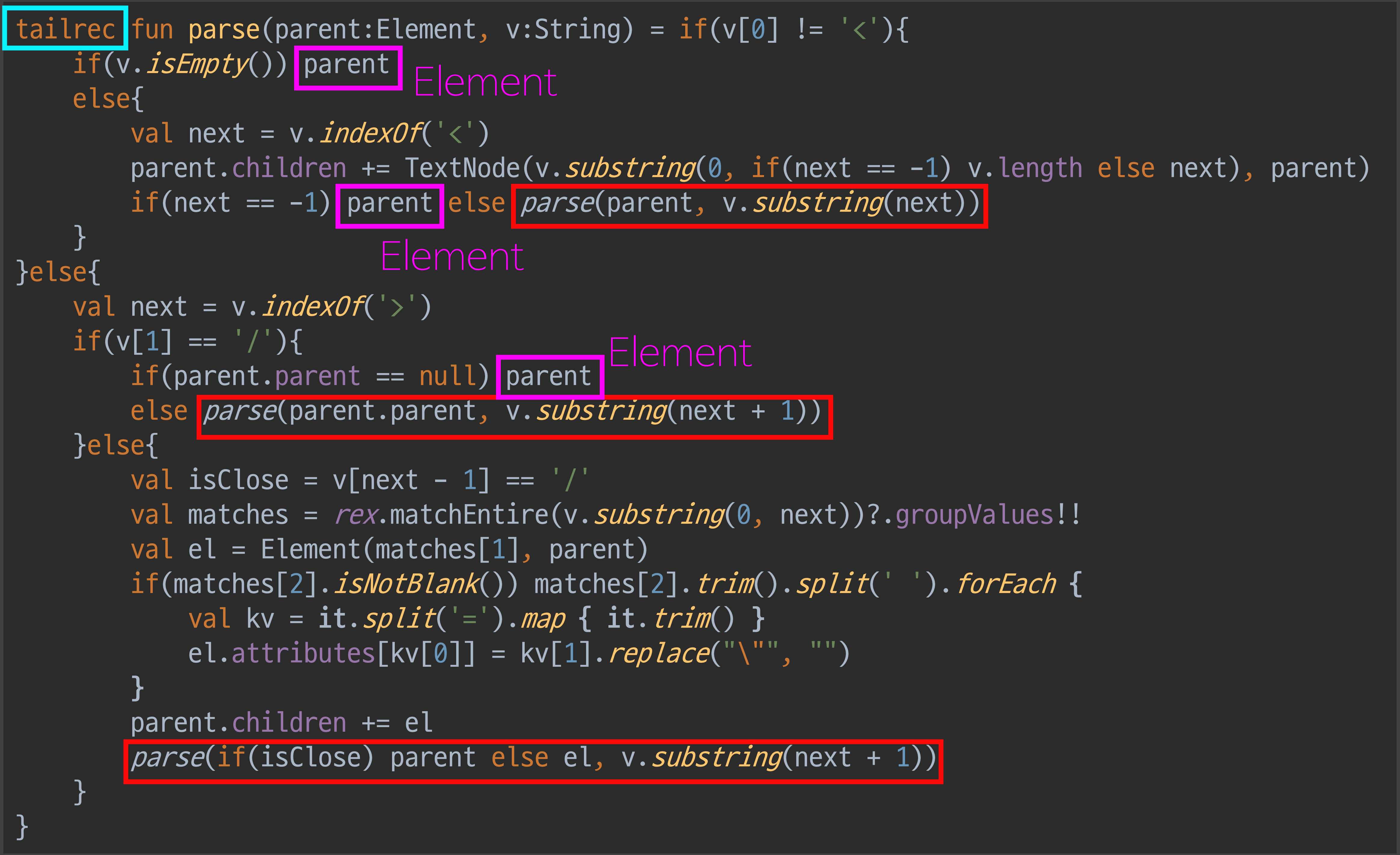
## Tail recursion & return Type



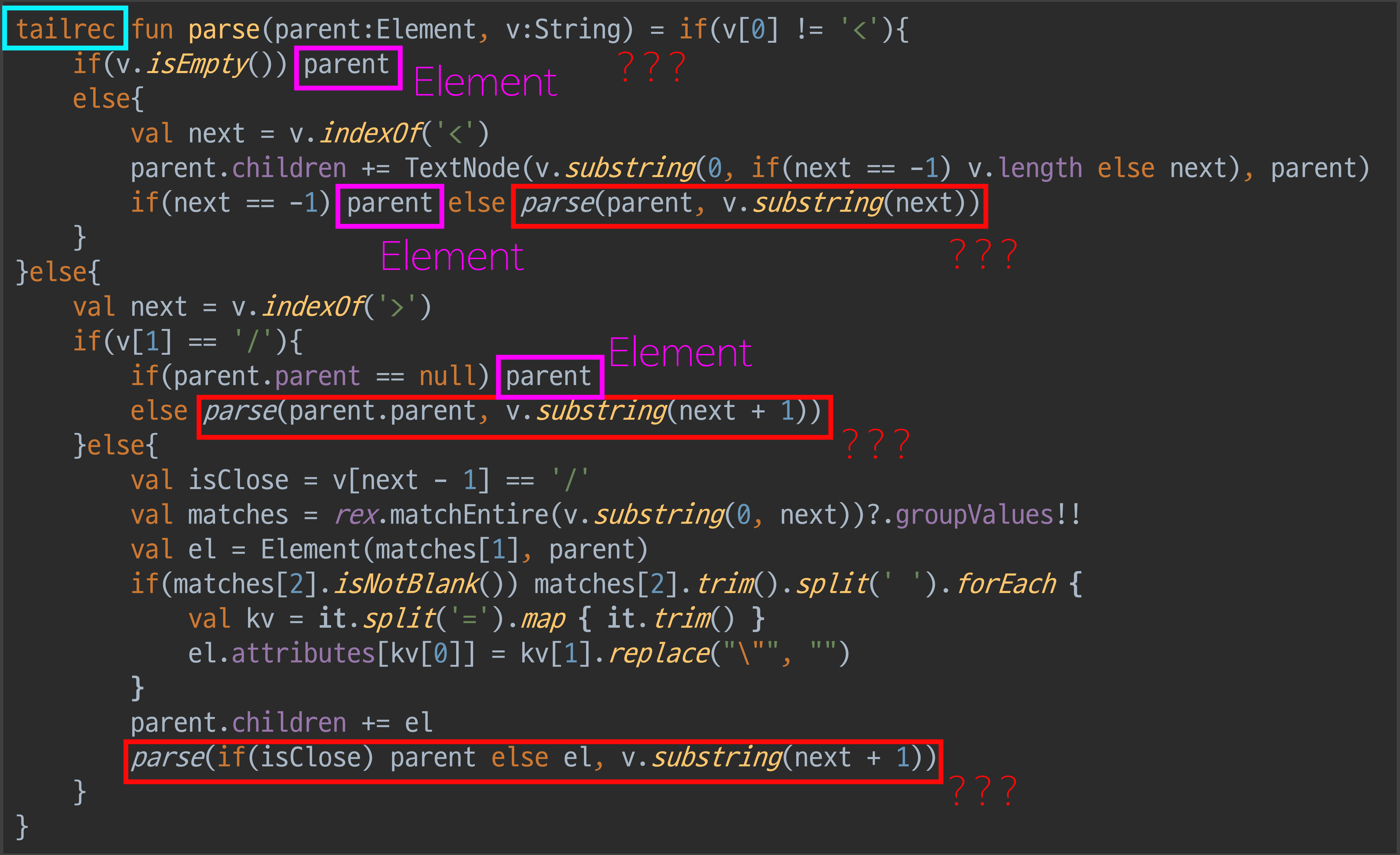
```
fun parse(parent:Element, v:String) = if(v[0] != '<'){
    if(v.isEmpty()) parent
    else{
        val next = v.indexOf('<')
        parent.children += TextNode(v.substring(0, if(next == -1) v.length else next), parent)
        if(next == -1) parent else parse(parent, v.substring(next))
    }
}else{
    val next = v.indexOf('>')
    if(v[1] == '/'){
        if(parent.parent == null) parent
        else parse(parent.parent, v.substring(next + 1))
    }else{
        val isClose = v[next - 1] == '/'
        val matches = rex.matchEntire(v.substring(0, next))?.groupValues!!
        val el = Element(matches[1], parent)
        if(matches[2].isNotBlank()) matches[2].trim().split(' ').forEach {
            val kv = it.split('=').map { it.trim() }
            el.attributes[kv[0]] = kv[1].replace("\"", "")
        }
        parent.children += el
        parse(if(isClose) parent else el, v.substring(next + 1))
    }
}
```

```kotlin
fun parse(parent:Element, v:String) = if(v[0] != '<'){
    if(v.isEmpty()) parent
    else{
        val next = v.indexOf('<')
        parent.children += TextNode(v.substring(0, if(next == -1) v.length else next), parent)
        if(next == -1) parent else parse(parent, v.substring(next))
    }
}else{
    val next = v.indexOf('>')
    if(v[1] == '/'){
        if(parent.parent == null) parent
        else parse(parent.parent, v.substring(next + 1))
    }else{
        val isClose = v[next - 1] == '/'
        val matches = rex.matchEntire(v.substring(0, next))?.groupValues!!
        val el = Element(matches[1], parent)
        if(matches[2].isNotBlank()) matches[2].trim().split(' ').forEach {
            val kv = it.split('=').map { it.trim() }
            el.attributes[kv[0]] = kv[1].replace("\"", "")
        }
        parent.children += el
        parse(if(isClose) parent else el, v.substring(next + 1))
    }
}
```

```kotlin
tailrec fun parse(parent:Element, v:String) = if(v[0] != '<'){
    if(v.isEmpty()) parent
    else{
        val next = v.indexOf('<')
        parent.children += TextNode(v.substring(0, if(next == -1) v.length else next), parent)
        if(next == -1) parent else parse(parent, v.substring(next))
    }
}else{
    val next = v.indexOf('>')
    if(v[1] == '/'){
        if(parent.parent == null) parent
        else parse(parent.parent, v.substring(next + 1))
    }else{
        val isClose = v[next - 1] == '/'
        val matches = rex.matchEntire(v.substring(0, next))?.groupValues!!
        val el = Element(matches[1], parent)
        if(matches[2].isNotBlank()) matches[2].trim().split(' ').forEach {
            val kv = it.split('=').map { it.trim() }
            el.attributes[kv[0]] = kv[1].replace("\"", "")
        }
        parent.children += el
        parse(if(isClose) parent else el, v.substring(next + 1))
    }
}
```



```kotlin
tailrec fun parse(parent:Element, v:String) = if(v[0] != '<'){
    if(v.isEmpty()) parent
    else{
        val next = v.indexOf('<')
        parent.children += TextNode(v.substring(0, if(next == -1) v.length else next), parent)
        if(next == -1) parent else parse(parent, v.substring(next))
    }
}else{
    val next = v.indexOf('>')
    if(v[1] == '/'){
        if(parent.parent == null) parent
        else parse(parent.parent, v.substring(next + 1))
    }else{
        val isClose = v[next - 1] == '/'
        val matches = rex.matchEntire(v.substring(0, next))?.groupValues!!
        val el = Element(matches[1], parent)
        if(matches[2].isNotBlank()) matches[2].trim().split(' ').forEach {
            val kv = it.split('=').map { it.trim() }
            el.attributes[kv[0]] = kv[1].replace("\"", "")
        }
        parent.children += el
        parse(if(isClose) parent else el, v.substring(next + 1))
    }
}
```



```kotlin
tailrec fun parse(parent:Element, v:String) = if(v[0] != '<'){
    if(v.isEmpty()) parent
    else{
        val next = v.indexOf('<')
        parent.children += TextNode(v.substring(0, if(next == -1) v.length else next), parent)
        if(next == -1) parent else parse(parent, v.substring(next))
    }
}else{
    val next = v.indexOf('>')
    if(v[1] == '/'){
        if(parent.parent == null) parent
        else parse(parent.parent, v.substring(next + 1))
    }else{
        val isClose = v[next - 1] == '/'
        val matches = rex.matchEntire(v.substring(0, next))?.groupValues!!
        val el = Element(matches[1], parent)
        if(matches[2].isNotBlank()) matches[2].trim().split(' ').forEach {
            val kv = it.split('=').map { it.trim() }
            el.attributes[kv[0]] = kv[1].replace("\"", "")
        }
        parent.children += el
        parse(if(isClose) parent else el, v.substring(next + 1))
    }
}
```

```kotlin
tailrec fun parse(parent:Element, v:String):Element = if(v[0] != '<'){
    if(v.isEmpty()) parent
    else{
        val next = v.indexOf('<')
        parent.children += TextNode(v.substring(0, if(next == -1) v.length else next), parent)
        if(next == -1) parent else parse(parent, v.substring(next))
    }
}else{
    val next = v.indexOf('>')
    if(v[1] == '/'){
        if(parent.parent == null) parent
        else parse(parent.parent, v.substring(next + 1))
    }else{
        val isClose = v[next - 1] == '/'
        val matches = rex.matchEntire(v.substring(0, next))?.groupValues!!
        val el = Element(matches[1], parent)
        if(matches[2].isNotBlank()) matches[2].trim().split(' ').forEach {
            val kv = it.split('=').map { it.trim() }
            el.attributes[kv[0]] = kv[1].replace("\"", "")
        }
        parent.children += el
        parse(if(isClose) parent else el, v.substring(next + 1))
    }
}
```

## print Element

# printElement

```kotlin
fun printElement(el:Element, indent:Int = 0){
  el.children.forEach {
    if(it is Element){

    }else if(it is TextNode){

    }
  }
}
```

# printElement

```
fun printElement(el:Element, indent:Int = 0){
  el.children.forEach {
    if(it is Element){

    }else if(it is TextNode){
      println("${"-".repeat(indent)}Text '${it.text}'")
    }
  }
}
```

# printElement

```kotlin
fun printElement(el:Element, indent:Int = 0){
  el.children.forEach {
    if(it is Element){
      println("${"-".repeat(indent)}Element ${it.tagName}")
    }else if(it is TextNode){
      println("${"-".repeat(indent)}Text '${it.text}'")
    }
  }
}
```

# printElement

```kotlin
fun printElement(el:Element, indent:Int = 0){
  el.children.forEach {
    if(it is Element){
      println("${"-".repeat(indent)}Element ${it.tagName}")
      if(it.attributes.isNotEmpty()){

      }
    }else if(it is TextNode){
      println("${"-".repeat(indent)}Text '${it.text}'")
    }
  }
}
```

# printElement

```
fun printElement(el:Element, indent:Int = 0){
  el.children.forEach {
    if(it is Element){
      println("${"-".repeat(indent)}Element ${it.tagName}")
      if(it.attributes.isNotEmpty()){
        println("${" ".repeat(indent + 2)}Attribute ${
          it.attributes.map{(k, v)->"$k = '$v'"}.joinToString(" ")
        }")
      }
      printElement(it, indent + 1)
    }else if(it is TextNode){
      println("${"-".repeat(indent)}Text '${it.text}'")
    }
  }
}
```

# printElement

```
printElement(parseHTML("""<div>
    test1
    <img/>
    test2
    <p a="3" b="abc">ptest</p>
</div>"""))
```

https://bit.ly/2Wr0BzV

```
Element div
-Text '
        test1
        '
-Element img
-Text '
        test2
        '
-Element p
   Attribute a = '3' b = 'abc'
--Text 'ptest'
-Text '
    '
```