Overloadable operators

TODO(rename this and all the refs to smth)

Some syntax forms in Kotlin are defined *by convention*, meaning that their semantics are defined through syntactic expansion of current syntax form into another syntax form. The expansion of a particular syntax form is a different piece of code usually defined in the terms of operator functions. Operator functions are function that are [declared][Function declaration] with a special keyword `operator` and are not different from normal functions when called normally, but allow themselves to be employed by syntactic expansion. Different platforms may add other criteria on whether a function may be considered a suitable candidate for operator convention.

Particular cases of definition by convention include:

- Arithmetic and comparison operators;
- Operator-form [assignments][Assignments];
- [For-loop statements][For-loop statement];
- [Delegated properties][Delegated property declaration];
- TODO(anything else?)

There are several common points among all the syntax forms defined using this mechanism:

- The expansions are hygenic: if they seemingly introduce new identifiers that were not present in original syntax, all such identifiers are not accessible outside the expansion and cannot clash with any other declarations in the program;
- The expressions captured by an expansion are using call-by-need evaluation strategy, meaning that they are evaluated only once during first usage specified in the expansion even if the expansion itself has more than one usage of such an expression;
- An expansion may lead to another expansion, following the same rules;
- All the new call expressions that are produced by expansion are only allowed to use operator functions.

For example, take the following declarations:

```
class A {
    operator fun inc(): A { ... }
}

object B {
    operator fun get(i: Int): A { ... }
    operator fun set(i: Int, value: A) { ... }
}

object C {
    operator fun get(i: Int): B { ... }
}
```

The expression `C[0][0]++` is expanded (see the [Expressions][expressions] section for details) using the following rules:

- First, the [increment operator][Postfix increment expression] is expanded, resulting in:

  ```
  C[0][0] = C[0][0].inc()
  ```

- Second, the [assigment][Assignments] to an indexing expression (produced by the previous expansion) is expanded, resulting in:

  ```
  C[0].set(C[0].get(0).inc())
  ```

- Third, the [indexing expression][Indexing expression] is expanded, resulting in:

  ```
  C.get(0).set(C.get(0).get(0).inc())
  ```

Although the resulting expression contains several invocations of the subexpression `C.get(0)`, it is evaluated only once, making this code roughly equivalent to:

```
val $tmp = C.get(0)
$tmp.set($tmp.get(0).inc())
```

TODO()