Project O
Object-oriented language

1. Неформальная спецификация языка O

Программа на языке O представляет собой последовательность объявлений классов (возможно, пустую). Точкой входа в программу по определению может служить конструктор любого ее класса. Имя начального класса задается при активации программы, вместе с аргументами конструктора (если они необходимы).

```
Program : { ClassDeclaration }
```

Семантика инициализации программы заключается в следующем. С помощью конструктора класса с параметрами, соответствующими переданным аргументам, создается неименованный объект данного класса, производится присваивание аргументов из командной строки запуска соответствующим параметрам конструктора, после чего управление передается телу конструктора. Завершение выполнения этого тела означает завершение работы всей программы.

```
ClassDeclaration
          : class ClassName [ extends ClassName ] is
              { MemberDeclaration }
            end
```

Понятие класса является основополагающим (и по сути, единственным) понятием системы типов языка O. Иными словами, единственный способ задания типов объектов (переменных) в языке O,- это определить некоторый класс. Таким образом, можно считать (аналогично другим объектно-ориентированным языкам), что «класс – это тип».

В простейшем случае класс представляет собой коллекцию логически связанных ресурсов – простых членов класса и методов класса. Простые члены класса в совокупности определяют текущее состояние («state») экземпляра класса; методы класса задают поведение («behavior») объектов данного класса. В классе могут также задаваться «специальные» методы, используемые при создании объектов (экземпляров) класса,- конструкторы.

Для классов определено отношение наследования. Это означает, что некоторый класс может включать, наряду с собственными (определенными в самом классе) ресурсами, также и ресурсы некоторого другого класса, который в данном случае называется базовым классом данного класса. Класс, наследующий свойства базового класса, называется производным классом. Тем самым можно строить иерархии классовых типов, в которых свойства производного класса представляют собой композицию собственных свойств и свойств базового класса.

Отношение наследования дает возможность работать с объектами производных классов как с некоторыми «расширениями» объектов базовых классов. В

частности, объекты типов базовых классов могут ссылаться на объекты, тип которых – производный класс.

Отношение наследования является транзитивным: если некоторый класс A является базовым для класса B, а класс B – базовый для класса C, то класс A считается также базовым и для класса C.

Отношение наследования определяет полиморфное поведение методов классов. Считается, что метод производного класса, совпадающий по своей сигнатуре с методом базового класса, перекрывает (замещает, «overrides») метод базового класса. Иными словами, в процессе выполнения программы выбор конкретного вызываемого метода определяется динамическим типом объекта.

```
ClassName : Identifier [ '[' ClassName ']' ]
```

В языке определено понятие обобщенных классов («generics»). Это означает, что в объявлении класса может быть задан один или несколько «абстрактных» типов (типов-параметров), которые могут использоваться для задания переменных и методов класса. Очевидным примером обобщенного класса служит массив, в котором тип элементов не фиксируется и задается как «абстрактный».

Использование обобщенного класса возможно после его настройки («instantiation»), в которой для всех абстрактных типов-параметров задаются имена реальных типов (в виде списка типов, разделяемых запятыми и заключенного в квадратные скобки). При обработке обобщенного класса компилятор реализует подстановку реальных типов вместо абстрактных и генерирует для результата подстановки реальный («настроенный») класс.

```
MemberDeclaration
          : VariableDeclaration
          | MethodDeclaration
          | ConstructorDeclaration
```

Как уже говорилось, класс определяется как совокупность переменных-членов класса, методов класса и специальных методов-конструкторов класса.

Считается, что переменные-члены класса доступны извне класса только по чтению; непосредственно изменить текущее значение переменной-члена (например, посредством присваивания) невозможно. Доступ к переменным-членам допускается только из методов класса (или из методов производных классов).

```
VariableDeclaration
          : var Identifier ':' Expression
```

Каждая сущность в программе должна быть объявлена. Относительный порядок объявления переменной и ее использования описывается далее. Синтаксис и семантика объявления переменных несколько отличается от привычного вида. Для объявления переменной необходимо задать два обязательных компонента: имя переменной и ее начальное значение. Язык O является типизированным, так

что тип объявляемой переменной однозначно определяется из типа инициализирующего выражения.

```
MethodDeclaration
        : method Identifier [ Parameters ] [ : Identifier ]
            is Body end
```

Методы класса в совокупности определяют все аспекты внешнего поведения класса и схему изменения его состояния. Доступ к переменным-членам класса по записи возможен только посредством методов класса.

Объявление метода начинается со служебного слова **method**, после которого задается идентификатор имени метода и, возможно, заключенный в круглые скобки список параметров метода. Если метод возвращает некоторое значение (и, тем самым, его вызов может входить в состав выражений), то после параметров метода должен следовать тип возвращаемого значения. Далее, после служебного слова **is** должна располагаться последовательность операторов тела метода. Тело метода завершается служебным словом **end**.

В классе может быть задано несколько одноименных методов, которые должны различаться количеством и типами параметров. Выбор конкретного вызываемого метода среди одноименных определяется компилятором на основе сопоставления аргументов вызова и параметров методов с данным именем.

```
Parameters
        : ( ParameterDeclaration { , ParameterDeclaration } )

ParameterDeclaration
        : Identifier : ClassName

Body : { VariableDeclaration | Statement }

ConstructorDeclaration
            : this [ Parameters ] is Body end

Statement   : Assignment
            | WhileLoop
            | IfStatement
            | ReturnStatement
```

В языке определен минимально необходимый для реального программирования набор операторов: присваивание, цикл, условный оператор и оператор возврата из метода.

```
Assignment : Identifier ':=' Expression
```

Присваивание реализует

```
WhileLoop   : while Expression loop Body end
```

В языке определена единственная (наиболее общая) форма оператора цикла: цикл по условию. Тело цикла выполняется, пока значение выражения после служебного слова **while** равняется **true**. Выражение в условии должно иметь тип

Boolean и вычисляется перед каждой итерацией цикла. Таким образом, цикл может выполняться ноль или более раз.

```
IfStatement : if Expression then Body [ else Body ] end
```

Условный оператор ставит выполнение некоторой поседовательности операторов в зависимость от выполнения некоторого условия. Условие задается в виде выражения, которое должно иметь тип Boolean, после служебного слова **if**. Если значение выражения есть **true**, то выполняется последовательность операторов, указанная после служебного слова **then**, в противном случае – последовательность операторов после служебного слова **else** (если эта часть оператора имеется).

Служебное слово **end**, завершающее условный оператор, используется для фиксации текстуального завершения оператора и предотвращает возможные неоднозначности в случае вложенных условных операторов.

```
ReturnStatement
          : return [ Expression ]
```

Оператор возврата реализует завершение выполнения метода и возврат управления в контекст вызывающего метода. Если метод, в теле которого встретился оператор возврата, задает тип возвращаемого значения, то оператор должен содержать выражение, определяющее возвращаемое значение.

```
Expression : Primary { '.' Identifier [ Arguments ] }

Arguments  : '(' Expression { ',' Expression } ')'
```

Выражения предназначены для задания способа вычисления значений. Структура выражений в языке О значительно проще, чем во многих других языках и содержит, по существу, две базовые конструкции – доступ к члену класса и вызов метода.

Доступ к члену класса (переменной или методу) реализуется посредством точечной нотации вида *имя-объекта.имя-члена*. Вызов метода формируется традиционным способом: имя метода (которое может, в свою очередь, задаваться в виде точеченой нотации), после которого в круглых скобках может следовать разделенный запятыми список аргументов метода. Так как вызов метода может возващать некоторый объект (переменную), конструкция вызова может служить левой частью конструкции доступа к члену. Тем самым, допускаются составные конструкции, образуемые суперпозицией вызовов и/или доступов к членам.

```
Primary    : IntegerLiteral
           | RealLiteral
           | BooleanLiteral
           | this
           | ClassName
```

Базовыми компонентами выражений служат первичные. Это либо литералы библиотечных типов (их лексический синтаксис представлен в отдельном

документе), либо служебное слово `this`, которое в теле метода обозначает тот объект (переменную), для которой вызывается метод.

Наконец, имя класса также считается первичным выражением. Имя класса представляет собой либо просто идентификатор – для случая простого (необобщенного) класса, либо идентификатор обобщенного класса, после которого в квадратных скобках указываются реальные типы, которыми настраивается обобщенный класс. Типы внутри квадратных скобок разделяются запятыми.

В языке O определено небольшое количество библиотечных (предопределенных) классов. Использование этих классов в программах не требует каких-либо явных способов их подключения. Иными словами, считается, что объявления библиотечных классов присутствуют в любой программе по умолчанию.

Детали реализации библиотечных классов в языке не определяются. Неформальное описание семантики этих классов дается ниже.

2. Стандартные классы языка O

Полная иерархия библиотечных классов выглядит следующим образом:

```
class Class is ... end
   class AnyValue extends Class is ... end
      class Integer extends AnyValue is ... end
      class Real extends AnyValue is ... end
      class Boolean extends AnyValue is ... end
   class AnyRef extends Class is ... end
      class Array extends AnyRef is ... end
      class List extends AnyRef is ... end
```

Библиотечные классы `Integer`, `Real` и `Boolean` реализуют привычные объекты целых, вещественных и булевских типов, соответственно. Литералы перечисленных типов могут задаваться традиционным способом, который описан в отдельном документе.

Для указанных библиотечных классов определены методы, реализующие традиционные для данных типов операции – арифметические, логические и операции отношения, а также операции преобразования типов. Все эти операции представляются в виде соответствующих методов. Таким образом, традиционная инфиксная запись выражений в языке не поддерживается. (Возможно, в последующих версиях языка привычная форма записи выражений будет реализована.)

```
class Integer extends AnyValue is
    // Constructors
    this(p: Integer)
    this(p: Real)

    // Features
    var Min : Integer
```

```
    var Max : Integer

    // Conversions
    method toReal : Real
    method toBoolean : Boolean

    // Unary operators
    method UnaryMinus : Integer

    // Integer binary arithmetics
    method Plus(p:Integer) : Integer
    method Plus(p:Real) : Real
    method Minus(p: Integer) : Integer
    method Minus(p: Real) : Real
    method Mult(p: Integer) : Integer
    method Mult(p: Real) : Real
    method Div(p: Integer) : Integer
    method Div(p: Real) : Real
    method Rem(p: Integer) : Integer

    // Relations
    method Less(p: Integer) : Boolean
    method Less(p: Real) : Boolean
    method LessEqual(p: Integer) : Boolean
    method LessEqual(p: Real) : Boolean
    method Greater(p: Integer) : Boolean
    method Greater(p: Real) : Boolean
    method GreaterEqual(p: Integer) : Boolean
    method GreaterEqual(p: Real) : Boolean
    method Equal(p: Integer) : Boolean
    method Equal(p: Real) : Boolean
end

class Real extends AnyValue is
    // Constructors
    this(p: Real)
    this(p: Integer)

    // Features
    var Min : Real
    var Max : Real
    var Epsilon : Real

    // Conversions
    method toInteger : Integer

    // Unary operators
    method UnaryMinus : Real

    // Real binary arithmetics
    method Plus(p:Real) : Real
    method Plus(p:Integer) : Real
    method Minus(p: Real) : Real
```

```
    method Minus(p: Integer) : Real
    method Mult(p: Real) : Real
    method Mult(p: Integer) : Real
    method Div(p: Integer) : Real
    method Div(p: Real) : Real
    method Rem(p: Integer) : Real

    // Relations
    method Less(p: Real) : Boolean
    method Less(p: Integer) : Boolean
    method LessEqual(p: Real) : Boolean
    method LessEqual(p: Integer) : Boolean
    method Greater(p: Real) : Boolean
    method Greater(p: Integer) : Boolean
    method GreaterEqual(p: Real) : Boolean
    method GreaterEqual(p: Integer) : Boolean
    method Equal(p: Real) : Boolean
    method Equal(p: Integer) : Boolean
end

class Boolean extends AnyValue is
    // Constructor
    this(Boolean)

    // Conversion
    method toInteger() : Integer

    // Boolean operators
    method Or(p: Boolean) : Boolean
    method And(p: Boolean) : Boolean
    method Xor(p: Boolean) : Boolean
    method Not : Boolean
end

class Array[T] extends AnyRef is
    // Constructor
    this(l: Integer)

    // Conversion
    method toList : List

    // Features
    method Length : Integer

    // Access to elements
    method get(i: Integer) : T
    method set(i: Integer, v: T)
end

class List[T] extends AnyRef is
    // Constructors
    this()
    this(p: T)
```

```
    this(p: T, count: Integer)

    // Operators on lists
    method append(v: T) : List
    method head() : T
    method tail() : List
    ....
end

method MaxInt(a: Array[Integer]) : Integer is
    var max : Integer.Min
    var i : Integer(1)
    while i.Less(a.Length) loop
        if a.get(i).Greater(max) then max := get(i) end
        i := i.Plus(1)
    end
    return max
end
```

3. Примеры конструкций языка O

```
var a : Array[Integer](10)
a.set(i) := 55
x := a.get(i.Plus(1))

var i is 1
while i.LessEqual(a.Size) loop
    x := a.get(i)
    a.set(i,x.Mult(x))
    i := i.Plus(1)
end

class C[T] is
    var m : T
end

var y : Array[Integer](10)
var k : List[Real]

var m : Integer(1)

var b : true
var b : Boolean(true)

var x : Base(1,2)
x := Derived(3)
```

4. Формальная грамматика языка O

```
Program    : { ClassDeclaration }

ClassDeclaration
           : class ClassName [ extends ClassName ] is
                { MemberDeclaration }
             end
```

```
ClassName  : Identifier [ '[' ClassName ']' ]

MemberDeclaration
           : VariableDeclaration
           | MethodDeclaration
           | ConstructorDeclaration

VariableDeclaration
           : var Identifier ':' Expression

MethodDeclaration
           : method Identifier [ Parameters ] [ : Identifier ]
                                 is Body end

Parameters : ( ParameterDeclaration { , ParameterDeclaration } )

ParameterDeclaration
           : Identifier : ClassName

Body       : { VariableDeclaration | Statement }

ConstructorDeclaration
           : this [ Parameters ] is Body end

Statement  : Assignment
           | WhileLoop
           | IfStatement
           | ReturnStatement

Assignment : Identifier ':=' Expression

WhileLoop  : while Expression loop Body end

IfStatement : if Expression then Body [ else Body ] end

ReturnStatement
           : return [ Expression ]

Expression : Primary { '.' Identifier [ Arguments ] }

Arguments  : '(' Expression { ',' Expression } ')'

Primary    : IntegerLiteral
           | RealLiteral
           | BooleanLiteral
           | this
           | ClassName
```