Project D
Dynamic Language

1. Краткая характеристика языка

- Object types are not specified and can change while program execution
- The language is **interpreted**
- Major notion: variable & literal (constant)
- Program structure: a sequence of declarations and statements
- Builtin types: built-in: integer, real, boolean, string
  User-defined types: array, tuple, function
- Implicit type conversions
- Statements: assignment, if/while, return, input/output

2. Неформальная спецификация языка

Программа представляет собой простую последовательность операторов. Операторы могут отделяться друг от друга символами «точка с запятой» или символами новой строки.

```
Program : { Statement ‘;’ }
```

Операторы выполняются в порядке один за другим, начиная с текстуально самого первого объявления.

Различаются две категории операторов: **объявление** переменной и собственно операторы. Объявление переменной вводит в текущую область действия новый именованный объект, который может хранить некоторое значение. Собственно операторы задают определенное действие, семантика которого может заключаться в вычислении нового значения некоторой переменной (assignment), либо в изменении последовательного порядка выполнения операторов, либо в выводе значения на консоль.

Объявление переменной может встретиться в любом месте программы. Переменная, введенная посредством объявления, считается активной в пределах ее области действия, начиная с точки ее объявления и до конца этой области действия. В пределах области действия допускается наличие только одной переменной с заданным именем.

Если в некоторой области действия содержатся другие (вложенные) области действия, то переменные, объявленные во вложенной области действия, могут иметь имена, совпадающие с именами из объемлющей области действия. В этом случае переменные из вложенной области действия скрывают переменные с такими же именами из объемлющей области действия.

```
Declaration   : var VariableDefinition
                      { ‘,’ VariableDefinition }

VariableDefinition : IDENT [ ':=' Expression ]
```

При объявлении переменной можно задать ее начальное значение. Начальное значение переменной может быть впоследствии изменено посредством оператора присваивания. Если начальное значение переменной не задано, то считается, что переменная имеет специальное значение **`empty`**. Значение **`empty`** не может выступать в качестве операнда операций; единственное действие над переменной с таким значением – проверка типа посредством операции **`is`**, см. далее.

Тип переменной при объявлении *не фиксируется*. Это означает, что переменная в пределах своей области действия может принимать значения любого типа из числа допустимых в языке (см. «Типы»). Для проверки типа текущего значения переменной имеются специальная унарная операция **`is`**.

В языке определен обычный набор общеупотребительных **операторов**: помимо объявлений, которые также имеют статус операторов, имеются присваивание, условный оператор, два вида операторов повторения, а также оператор возврата значения из функции и вывод значений на печать.

```
Statement     : Declaration
              | Assignment
              | If
              | Loop
              | Return
              | Print

Assignment    : Reference ':=' Expression

If            : if Expression then Body [ else Body ] end

Loop          : while Expression loop Body end
              | for [ IDENT in ] [ Expression '..' Expression ]
                  loop Body end

Return        : return [ Expression ]

Print         : print Expression { ',' Expression }
```

Выражения – синтаксические конструкции, задающие алгоритм вычисления новых значений. Структура выражений традиционна и включает несколько видов операндов и набор инфиксных и префиксных операций над операндами.

```
Expression    : Relation { ( or | and | xor ) Relation }

Relation      : Factor [ ( '<'|'<='|'>'|'>='|'='|'/=' ) Factor ]

Factor        : Term { [ '+' | '-' ] Term }

Term          : Unary { ( '*' | '/' ) Unary }

Unary         : Reference
              | Reference is TypeIndicator
              | [ '+' | '-' | not ] Primary
```

```
Primary       : Literal
              | readInt | readReal | readString
              | FunctionLiteral
              | '(' Expression ')'

TypeIndicator : int | real | bool | string
              | empty     // no type
              | '[' ']'   // vector type
              | '{' '}'   // tuple type
              | func      // functional type
```

Функции в языке D считаются литералами (наравне с целыми/вещественными значениями) в том смысле, что они рассматриваются как константные значения, которые можно присваивать переменным и передавать в качестве аргументов в другие функции. Единственная операция, определенная для функций, – вызов.

```
FunctionLiteral : func [ '(' IDENT { ',' IDENT } ')' ] FunBody

FunBody : is Body end
        | => Expression
```

Правило `Reference` определяет *ссылки* – конструкции, которые реализуют элементарные действия с переменными: взятие элемента массива, вызов функции и доступ к элементам объектов. Ссылки могут служить операндами выражений, а также (что отличает их от других операндов – констант, подвыражений и др.) выступать в качестве получателей значений в операторах присваивания.

```
Reference : IDENT
          | Reference '[' Expression ']'
          | Reference '(' Expression { ',' Expression } ')'
          | Reference '.' IDENT
```

Типы и литералы. В языке определены значения четырех простых типов, двух составных типов и одного специального типа. Простые типы: целый, вещественный, строковый и булевский. Литералы целого типа записываются в виде последовательности десятичных цифр. Вещественные литералы образуются в виде двух последовательностей десятичных цифр, разделенных символом «точка». Первая последовательность обозначает целую часть вещественного, вторая последовательность – его дробную часть. Строки задаются в виде последовательности произвольных символов, заключенных в одинарные или двойные кавычки. Значения булевского типа представляются двумя служебными словами `true` (истина) и `false` (ложь).

```
Literal    : INTEGER
           | REAL
           | STRING
           | Boolean
           | Tuple      // { a := 5, b := “sss”, 12.34 }
           | Array      // [ 1, 2, 3 ]
           | empty
```

```
Boolean    : true | false
```

Кроме простых типов, язык включает два составных типа: массивы (arrays) и кортежи (tuples).

Массив – это линейная композиция значений *любых типов*. Размер массива (число элементов) не фиксируется и может динамически изменяться. Массив записывается в виде списка значений элементов, разделяемых запятыми и заключенного в угловые скобки.

```
Array : '[' [ Expression { ',' Expression } ] ']'
```

Доступ к элементам массива производится обычным образом: после имени массива в квадратных скобках задается порядковый номер элемента в виде целочисленного выражения. Нумерация элементов массива начинается с единицы.

Заметим, что массив – это «настоящий» ассоциативный массив с ключами, представляющими целые значения. Значения соседних ключей в массиве не обязательно различаются на единицу. Так, возможна, например, следующая последовательность операторов:

```
var t := [];   // объявление пустого массива
t[10] := 25;
t[100] := func(x)=>x+1;
t[1000] := {a:=1,b:=2.7};
```

Кортеж – это совокупность поименованных значений произвольных типов. Имена элементов уникальны в пределах кортежа. Кортеж задается в виде разделенного запятыми списка пар вида «имя – значение». Имя и значение разделяются символом присваивания. Весь список заключается в фигурные скобки.

```
Tuple : '{' TupleElement { ',' TupleElement } '}'

TupleElement : [ Identifier ':=' ] Expression
```

Структура кортежей, в отличие от массивов, не может модифицироваться; единственный способ изменить состав элементов кортежа – добавить к нему другой кортеж:

```
var t := {a:=1, b:=2, c};
t := t + {d:=3};   // now t is {a=1, b=2, c, d=3}
```

Доступ к элементам кортежа производится посредством точечной нотации, в которой задается имя переменной-кортежа и имя элемента кортежа:

```
var x := t.b;     // now x is 2
```

Допускается задание неименованных элементов кортежа. Именованные и неименованные элементы могут присутствовать в кортеже в любом порядке. Доступ к неименованным элементам кортежа осуществляется посредством

точечной нотации, где элемент кортежа идентифицируется его порядковым номером. Первый элемент всегда имеет номер 1. Пример:

```
var y1 := t.1   // now y1 is 1
var y2 := t.3   // now y2 has the value of c
```

Операции над значениями. Семантика операций в целом типичная для языков программирования и зависит от типа операндов (операнда). Так как язык динамический, то контроль типов операндов и выбор соответствующего алгоритма операции производится динамически, во время исполнения программы.

Ниже приводится таблица с кратким описанием операций.

Сложение +

Целое + Целое -> Целое
Целое + Вещественное -> Вещественное
Вещественное + Целое -> Вещественное
Вещественное + Вещественное -> Вещественное
Строка + Строка -> Строка (конкатенация строк)
Кортеж + Кортеж -> Кортеж (конкатенация кортежей)
Массив + Массив -> Массив (конкатенация массивов
Прочие типы операндов не допускаются

Вычитание -

Целое - Целое -> Целое
Целое - Вещественное -> Вещественное
Вещественное - Целое -> Вещественное
Вещественное - Вещественное -> Вещественное
Прочие типы операндов не допускаются

Умножение *

Целое * Целое -> Целое
Целое * Вещественное -> Вещественное
Вещественное * Целое -> Вещественное
Вещественное * Вещественное -> Вещественное
Прочие типы операндов не допускаются

Деление /

Целое / Целое -> Целое (деление нацело с отбрасыванием др. части)
Целое / Вещественное -> Вещественное
Вещественное / Целое -> Вещественное
Вещественное / Вещественное -> Вещественное
Прочие типы операндов не допускаются

Сравнения <, >, <=, >=, =, /=

Целое *op* Целое -> Булевское
Целое *op* Вещественное -> Булевское
Вещественное *op* Целое -> Булевское
Вещественное *op* Вещественное -> Булевское
Прочие типы операндов не допускаются

Унарное сложение и вычитание + -

op Целое -> Целое
op Вещественное -> Вещественное
Прочие типы операндов не допускаются

Логические операции and, or xor, not

Оба операнда должны быть булевскими. Результат операций всегда булевский.
Прочие типы операндов не допускаются.

3. Грамматика языка

```
Program      : { Declaration }
Declaration  : var Identifier [ := Expression ] ;
Expression   : Relation [ ( and | or | xor ) Relation ]
Relation     : Factor [ ( < | <= | > | >= | = | /= ) Factor ]
Factor       : Term { ( + | - ) Term }
Term         : Unary { ( * | / ) Unary }
Unary        : [ + | - | not ] Primary [ is TypeIndicator ]
             | Literal
             | ( Expression )
Primary      : Identifier { Tail }
             | readInt | readReal | readString
Tail         : . IntegerLiteral   // access to unnamed tuple element
             | . Identifier       // access to named tuple element
             | [ Expression ]     // access to array element
             | ( Expression { , Expression } )  // function call
Statement    : { Assignment | Print | Return | If | Loop }
Assignment   : Primary := Expression
Print        : print Expression { , Expression }
Return       : return [ Expression ]
If           : if Expression then Body [ else Body ] end
Loop         : while Expression LoopBody
             | for Identifier in TypeIndicator LoopBody
LoopBody     : loop Body end
```

```
TypeIndicator
              : int | real | bool | string
              | empty            // no type
              | [ ]              // vector type
              | { }              // tuple type
              | func             // functional type
              | Expression .. Expression

Literal       : IntegerLiteral    // 1, 12345, 777
              | RealLiteral       // 1.23
              | BooleanLiteral    // true or false
              | StringLiteral     // "string", 'string‘
              | ArrayLiteral
              | TupleLiteral

ArrayLiteral : [ [ Expression { , Expression } ] ]

TupleLiteral : { [ [ Identifier := ] Expression
                         { , [ Identifier := ] Expression } ] }

FunctionLiteral
              : func [ Parameters ] FunBody

Parameters    : ( Identifier { , Identifier } )

FunBody       : is Body end
              | => Expression

Body          : { Declaration | Statement | Expression }
```

4. Семантика базовых операций

Sign	Operand1	Operand2	Result	Semantics
+	Integer	Integer	Integer	Algebraic addition
	Integer	Real	Real	
	Real	Integer	Real	
	Real	Real	Real	
	Vector	Integer	Vector	Adding element to vector
	Vector	Real	Vector	
	Vector	Vector	Vector	Joining vectors
	List	Integer	List	Appending element to list
	List	Real	List	
	List	List	List	Joining lists
+=	Integer	Integer	Integer	
	Tuple	Tuple	Tuple	Memberwise addition
-	Integer	Integer	Integer	Algebraic subtraction
	Integer	Real	Real	
	Real	Integer	Real	
	Real	Real	Real	

5. Структура D-процессора