Project I
Imperative Language

1. Неформальное описание входного языка

Текст программы представляет собой последовательность символов в кодировке Unicode, структурно организованный согласно синтаксическим правилам, представленным ниже. Текст программы располагается в одном файле в файловой системе инструментальной платформы.

```
Program : { SimpleDeclaration | RoutineDeclaration }
```

Программа представляет собой последовательность объявлений переменных, типов и подпрограмм (возможно, пустую). Все ресурсы (переменные, типы и подпрограммы), используемые в программе, должны быть объявлены текстуально до момента их первого использования. Подключение к программе внешних ресурсов («импорт») не предусматривается.

Выполнение программы начинается с запуска некоторой подпрограммы из числа объявленных. Имя подпрограммы и ее аргументы (если они требуются) передаются программе из внешней среды при ее запуске. Тем самым, выполнение программы может начаться с любой ее подпрограммы.

Объявления отделяются друг от друга либо символами «новая строка», либо символами «точка с запятой». В целях большей компактности синтаксических правил символы-разделители в них не указываются.

В языке определены всего три категории сущностей: переменные, типы и подпрограммы. Это делает язык компактным, обозримым, легким для изучения и использования.

Область действия имени сущности, введенной объявлением, распространяется от точки ее объявления до текстуального конца области, в пределах которой переменная объявлена (то есть, до конца области, описываемой синтаксическим правилом Body, либо, для сущностей, объявленных на самом внешнем уровне, до физического конца текста программы). Если в пределах этой области задана некоторая вложенная область действия, то объявление сущности во вложенной области скрывает объявления одноименных сущностей в объемлющих блоках.

Это правило действует единообразно для имен всех сущностей – переменных, подпрограмм и типов.

```
SimpleDeclaration : VariableDeclaration | TypeDeclaration
```

Простые объявления включают объявления переменных и типов. Такие объявления могут задаваться в любой области действия (в том числе, в телах подпрограмм и составных операторов). В этом заключается отличие простых объявлений от объявлений подпрограмм, которые могут задаваться только на самом внешнем уровне вложенности. Иными словами, вложенность подпрограмм не допускается.

```
VariableDeclaration
            : var Identifier ':' Type [ is Expression ]
            | var Identifier           is Expression
```

Все переменные, используемые в программе, должны быть объявлены. Объявление переменной должно текстуально предшествовать ее первому использованию.

Объявление переменной включает a) имя переменной (идентификатор), по которому будет осуществляться доступ к значению, хранимому в ней, b) тип переменной (который определяет множество допустимых значений и множество операций над значениями), а также c) начальное значение переменной.

Тип переменной в объявлении может быть опущен, если в нем задано начальное значение. В этом случае тип может быть однозначно выведен из типа выражения, задающего начальное значение (type inference). Если начальное значение в объявлении не задано, то указание типа переменной является обязательным.

```
TypeDeclaration : type Identifier is Type
```

Объявление типа служит для введения краткого эквивалента (в виде идентификатора) некоторого типа. После объявления типа идентификатор, заданный после служебного слова **type**, считается эквивалентом типу, заданному после служебного слова **is**, и может использоваться везде в своей области действия, там, где требуется задание типа.

```
Type : PrimitiveType | UserType | Identifier
```

В языке определены две категории типов: предопределенные (базовые) типы и типы, определяемые пользователем. Для целей большей компактности и наглядности сложные типы могут обозначаться идентификаторами, введенными ранее посредством объявлений типов.

```
PrimitiveType : integer | real | boolean
```

В языке имеется три предопределенных типа: целочисленный, вещественный и булевский. Предопределенные типы вводятся посредством соответствующих служебных слов. Целочисленный тип включает множество (положительных и отрицательных) целых чисел. Вещественный тип включает множество (положительных и отрицательных) вещественных чисел.

Внутреннее представление значений, а также минимальное и максимальное значения целого и вещественного типов в языке не фиксируются. Лексические правила задания констант целого и вещественного типов определены единообразно для всех языков проекта DOFIN и описываются в отдельном документе.

Булевский тип включает два логических значения истинности – «истина» и «ложь», которые обозначаются посредством служебных слов **true** и **false**, соответственно.

```
UserType : ArrayType | RecordType
```

В языке определены две категории пользовательских типов: записи и массивы.

Пользовательские типы относятся к категории «ссылочных» типов; это означает, что переменные этих типов содержат не собственно значения, а ссылки на сам объект-агрегат. Тем самым, такие значения являются, фактически, «представителями» собственно агрегатов. Из этого, в частности, следует, что присваивание массива или записи некоторому другому массиву или записи приводит к копированию не самого агрегатного значения, а только ссылки на него.

```
RecordType : record { VariableDeclaration } end
```

Тип записи предназначен для группирования логически связанных переменных в единую конструкцию. В записи допускается произвольное число переменных-членов. Переменные-члены записи могут иметь произвольные типы; допускаются, например, массивы и (вложенные) записи. Доступ к переменным-членам записи производися посредством точечной нотации (см. ниже).

```
ArrayType : array '[' Expression ']' Type
```

Тип массива предназначен для задания групп переменных одного типа. Доступ к значениям-элементам массива реализуется посредством задания имени массива и индекса (порядкового номера) элемента внутри массива. Индекс элемента в массиве может задаваться динамически; тем самым допускается групповая работа с элементами массива, например, посредством операторов циклов (см. ниже).

Элементы массива нумеруются целыми числами. Первый элемент массива всегда имеет индекс, равный единице. При задании типа массива задается его размер, который должен быть целым числом. Выражение, задающее размер массива, должно представлять собой «константное выражение»: оно вычисляется до выполнения программы (на этапе ее компиляции).

Размер массива в объявлении может быть опущен. Тип массива без задания размера может использоваться только в качестве типа параметра подпрограммы. В этом случае информация о «реальном» размере массива, переданного в качестве аргумента, доступна посредством точечной нотации специального вида (см. ниже)

```
Statement  : Assignment | RoutineCall
           | WhileLoop | ForLoop | IfStatement
```

В языке определен типичный набор минимально необходимых операторов, реализующих содержательные действия над объектами программы (переменными).

```
Assignment : ModifiablePrimary ':=' Expression
```

Присваивание осуществляет вычисление выражения, заданного в правой части оператора, и последующее копирование значения в переменную, задаваемую в

левой части. Типы левой и правой частей присваивания должны быть совместимыми. Для переменных предопределенных типов совместимость определяется наиболее простым и очевидным из возможных способов:

Тип левой части	Тип правой части	Семантика присваивания
integer	integer	прямое копирование значения
integer	real	округление вещественного до ближайшего целого
integer	boolean	значение **true** преобразуется в целое 1, значение **false** – в 0
real	real	прямое копирование значения
real	integer	целое обобщается до вещественного
real	boolean	значение **true** преобразуется в вещественное 1.0, **false** – в 0.0
boolean	boolean	прямое копирование значения
boolean	integer	если целое есть 1, то оно преобразуется в **true**, если оно равно 0, то преобразуется в **false**; в противном случае присваивание считается недопустимым.
boolean	real	типы считаются несовместимыми; присваивание недопустимо.

Для переменных пользовательских типов (записей и массивов) совместимость означает полную именную идентичность типов левых и правых частей присваивания. Иными словами, левые и правые части присваивания должны иметь тип, определенный ранее посредством одного и того же объявления типа.

Следует иметь в виду, что пользовательские типы являются ссылочным. Это означает, что при присваивании копируется ссылка на массив/запись, а не сам агрегат. Тем самым, возможны ситуации, когда более одной переменной пользовательского типа именуют (ссылаются на) один и тот же агрегат.

Заметим, что передача аргументов в подпрограммы, а также возврат значения также считаются присваиванием, и в этих случаях применяются в точности те же самые правила.

```
RoutineCall
      : Identifier [ '(' Expression { ',' Expression } ')' ]
```

Конструкция `RoutineCall` задает вызов (активацию) подпрограммы, заданную идентификатором. Если подпрограмма объявлена с параметрами, то конструкция вызова должна включать список аргументов – выражений, разделенных запятыми и заключенный в круглые скобки. Число аргументов должно точно соответствовать количеству параметров в объявлении подпрограммы, а типы аргументов должны быть совместимыми с типами соответствующих параметров, см. «Assignment».

В конструкции вызова допускается задание имени подпрограммы, возвращающей значение («функции»). В этом случае значение, возвращаемое функцией, использоваться не будет (пропадает).

Заметим, что конструкция вызова подпрограммы синтаксически подобна и семантически идентична конструкции `ModifiablePrimary` в варианте с круглыми скобками.

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

Цикл с предусловием задает повторяющееся выполнение операторов тела. Выражение, заданное в заголовке цикла, должно иметь тип, совместимый с типом **`boolean`**.

Выполнение оператора цикла с предусловием заключается в вычислении выражения и проверке полученного значения. Если это значение есть **`true`**, то производится выполнение операторов и объявлений, заданных в теле цикла, после чего производится повторное вычисление выражения. Если первое или очередное вычисление дает значение **`false`**, то выполнение оператора цикла завершается. Таким образом, тело цикла может быть выполнено ноль или более раз.

```
ForLoop : for Identifier in [ reverse ] Range loop Body end

Range   : Expression '..' Expression
```

Цикл с диапазоном (for-цикл) реализует повторяющееся выполнение операторов тела. На каждой итерации цикла значение переменной, обозначенной идентификатором из заголовка, последовательно принимает целочисленные значения из диапазона, заданного конструкцией `Range`. В случае отсутствия служебного слова **`reverse`** значения изменяются в сторону увеличения на единицу; в противном случае, значения последовательно уменьшаются на единицу.

Переменная цикла считается неявно объявленной как имеющая тип **`integer`**. Иными словами, нет необходимости в специальном объявлении этой переменной. Областью действия переменной цикла служит тело цикла; тем самым, эта переменная, согласно общим правилам блочности, скрывает любое объявление одноименной переменной в области действия, заключающей цикл. После завершения выполнения цикла переменная считается необъявленной. Внутри тела цикла присваивания переменной цикла недопустимы.

Типы выражений в диапазоне `Range` должны быть совместимы с целым типом. Если значение первого выражения в диапазоне больше значения второго выражения, то тело цикла не выполнится ни одного раза. Значения выражений из диапазона вычисляются один раз перед выполнением самой первой итерации.

С помощью циклов с диапазоном удобно строить программные конструкции, работающие с элементами массивов.

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

Условный оператор задает зависимость выполнения некоторой последовательности операторов от текущего значения выражения. Выполнение начинается с вычисления значения выражения из заголовка условного. Тип выражения должен быть совместим с булевским типом. Если значение выражения есть **true**, то выполняется последовательность операторов, следующая после служебного слова **then**. В противном случае (если значение выражения есть **false**) выполнются операторы после служебного слова **else**, если эта часть присутствует в операторе.

```
RoutineDeclaration
    : routine Identifier [ Parameters ] [ ':' Type ]
      [ is Body end ]

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

ParameterDeclaration
             : Identifier : Type

Body         : { SimpleDeclaration | Statement }
```

Выражения строятся по обычным правилам согласно старшинству (приоритетам) операций.

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

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

Simple   : Factor { ( '*' | '/' | '%' ) Factor }

Factor   : Summand { ( '+' | '-' ) Summand }

Summand  : Primary | '(' Expression ')'

Primary  : [ Sign | not ] IntegerLiteral
         | [ Sign ] RealLiteral
         | true
         | false
         | ModifiablePrimary
         | RoutineCall

Sign     : '+' | '-'
```

Правило `ModifiablePrimary` представляет возможные конструкции левой части присваиваний. Помимо идентификаторов простых переменных, конструкция может задавать индекс элемента массива, а также имя элемента записи.

```
ModifiablePrimary
             : Identifier { '.' Identifier
                          | '[' Expression ']'
                          }
```

2. Формальная грамматика языка

```
Program      : { SimpleDeclaration | RoutineDeclaration }

SimpleDeclaration
             : VariableDeclaration | TypeDeclaration

VariableDeclaration
             : var Identifier : Type [ is Expression ]
             | var Identifier          is Expression

TypeDeclaration
             : type Identifier is Type

RoutineDeclaration
             : routine Identifier [ Parameters ] [ : Type ]
                                                 [ is Body end ]

Parameters   : ( ParameterDeclaration { , ParameterDeclaration } )

ParameterDeclaration
             : Identifier : Identifier

Type         : PrimitiveType | ArrayType | RecordType | Identifier

PrimitiveType: integer | real | boolean

RecordType   : record { VariableDeclaration } end

ArrayType    : array [ Expression ] Type

Body         : { SimpleDeclaration | Statement }

Statement    : Assignment | RoutineCall | WhileLoop | ForLoop
             | ForeachLoop | IfStatement

Assignment   : ModifiablePrimary := Expression

RoutineCall  : Identifier [ ( Expression { , Expression } ) ]

WhileLoop    : while Expression loop Body end

ForLoop      : for Identifier Range loop Body end

Range        : in [ reverse ] Expression .. Expression

ForeachLoop  : foreach Identifier from ModifiablePrimary loop Body end

IfStatement  : if Expression then Body [ else Body ] end

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

Relation     : Simple [ ( < | <= | > | >= | = | /= ) Simple ]
```

```
Simple          : Factor { ( * | / | % ) Factor }

Factor          : Summand { ( + | - ) Summand }

Summand         : Primary | ( Expression )

Primary         : IntegralLiteral | RealLiteral | true | false
                | ModifiablePrimary

ModifiablePrimary
                : Identifier { . Identifier | [ Expression ]
                | ( Expression { , Expression } ) ]
```