Сообщение о языке программирования Лиса

Сарыкова А.А.

16 марта 2019г.

Оглавление

Глава 1. Язык

Глава 2. Лексическая структура

В данной главе описывается лексическая структура языка.

2.1. Обозначения

Для описания как лексической структуры, так и для описания синтаксиса используется запись в виде расширенных формул Бэкуса–Наура (РБНФ). В используемом варианте РБНФ формула состоит из двух частей: первая часть содержит имя определяемого понятия, выделенное зелёным цветом, а затем, после метасимвола →, идёт вторая часть, содержащая определение понятия. В этой второй части допускается использование регулярных выражений над терминалами и нетерминалами, а в этих регулярных выражениях допускаются следующие операторы:

1) | — бинарный инфиксный оператор, означает „или“;
2) * — унарный постфиксный оператор, означает „может повторяться любое число раз, в том числе ни разу“;
3) + — унарный постфиксный оператор, означает „может повторяться любое число раз, но хотя бы один раз“;
4) ? — унарный постфиксный оператор, означает „может присутствовать, а может отсутствовать“;
5) () — эти скобки группируют конструкции.

При этом сначала выполняется то, что в скобках; затем, слева направо, — унарные операторы; и в конце — бинарный инфиксный оператор, тоже слева направо.

2.2. Кодировка исходного текста программы

Каждый файл исходного текста должен быть в кодировке UTF-8 (без BOM).

2.3. Ключевые слова

В языке есть зарезервированные слова, которые не могут использоваться в качестве имён каких–либо модулей, функций, переменных, констант, типов, и компонент типов. Данные зарезервированные слова далее будут называться ключевыми словами. Ниже приведён исчерпывающий список ключевых слов.

бескон	**возврат**	**компл128**	**перем**
бз	**выбор**	**конст**	**перечисление**
бз8	**выдел**	**логик**	**повт**
бз16	**выход**	**логик8**	**пока**
бз32	**глав**	**логик16**	**разбор**
бз64	**для**	**логик32**	**симв**
бз128	**если**	**логик64**	**строка**
бол	**иначе**	**ложь**	**структ**
в	**иначеесли**	**мал**	**ссылка**
вещ	**истина**	**массив**	**тип**
вещ32	**компл**	**множество**	**то**
вещ64	**компл32**	**модуль**	**фун**
вещ80	**компл64**	**ничего**	**чистая**
вещ128	**компл80**	**очищ**	**цел**

цел8 **цел32** **цел128**
цел16 **цел64**

2.4. Идентификаторы

Идентификатор — это последовательность букв, десятичных цифр, и знаков подчёркивания, которая должна начинаться либо с буквы, либо со знака подчёркивания. Под буквой понимается любая латинская буква или кириллическая буква. Идентификатор не может совпадать ни с одним ключевым словом.

Примеры идентификаторов: `крокодил`, `Гена`, `_s1024`, `Баба_Яга`, `Чебурашка`.

2.5. Числа

2.5.1. Целые числа

Целые числа могут быть двоичными, восьмеричными, шестнадцатиричными, и десятичными. Ниже приведён синтаксис записи целых чисел:

двоичное_целое→0(b|B)(0|1)('?(0|1))*
восьмеричное_целое→0(o|O)(0|1|2|3|4|5|6|7)('?(0|1|2|3|4|5|6|7))*
десятичное_целое→(0|1|2|3|4|5|6|7|8|9)('?(0|1|2|3|4|5|6|7|8|9))*
шестнадцатиричное_целое→0(x|X)шестнадцатиричная_цифра('?шестнадцатиричная_цифра)*
шестнадцатиричная_цифра→0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|a|b|c|d|e|f

2.5.2. Вещественные числа

Вещественные числа выглядят следующим образом:

вещественное→целая_часть(.дробная_часть)?((E|e)(+|-)?порядок)?суффикс_точности?
целая_часть→десятичное_целое
дробная_часть→десятичное_целое
порядок→десятичное_целое
суффикс_точности→f|d|x|q

Необязательный суффикс_точности указывает, какую точность имеет вещественный литерал: одинарную, если указано f; двойную, если указано d; расширенную (т.е. 80 разрядов), если указано x; и четырёхкратную (т.е. в числе 128 разрядов), если указано q. Если не указано никакого суффикса, то вещественный литерал рассматривается как имеющий двойную точность.

2.5.3. Комплексные числа

Поддерживаются только чисто мнимые комплексные литералы. Для этого нужно сразу после записи вещественного литерала записать букву i. При этом точность комплексного литерала — такая же, как и точность вещественного литерала, идущего перед i. Если же нужно записать комплексное число, у которого и вещественная, и мнимая части не равны нулю, то нужно писать соединять их знаками + или -. Например, если нужно записать комплексное число $-3.876 - 45.67i$, обе компоненты которого имеют четырёхкратную точность, то нужно писать `-3.876q-45.67qi`.

2.6. Символьные литералы

Символьный литерал можно задать двумя способами. Во–первых, можно просто заключить нужный символ в одинарные кавычки. Если при этом нужным символом является сам символ одинарной кавычки, то этот символ нужно удвоить. Примеры: `'я'`, `''''`. В первом случае определяется символ, представляющий русскую букву „я“, а во втором случае — символ, представляющий одинарную кавычку. Во–вторых, символьный литерал можно задать кодом соответствующего символа. Для этого нужно записать знак $, сразу после которого указать код символа, двоичный, восьмеричный, десятичный, или шестнадцатеричный.

2.7. Строковые литералы

Строковый литерал можно задать тремя способами: либо как цепочку символьных литералов, заданных своими кодами; либо как цепочку символов, заключённых в двойные кавычки (если нужно внутри указать символ двойной кавычки, то его нужно удвоить), либо как чередование того и другого.

Примеры строковых литералов:

"— пустая строка

```
"Хорошо живёт на свете Винни-Пух!"$13$10

"У попа была собака, он её любил.
Она съела кусок мяса, -- он её убил.
В землю закопал и надпись надписал: ..."

"Привет лунатикам!"$0
```

2.8. Знаки операций и разделители

В языке имеются следующие знаки операций и разделители:

```
[       .       |       {       ..      >>      ++<     ^=      ^^=     **.=
]       :       &       }       ?.      ##      --<     ||.     **=     !||=
(       ;       <       +|      <=      =       +=      &&.     <<=     !&&=
)       ==      >       ++      >=      @@      -=      +.=     >>=     !||.=
<-      #       ?       --      !=      ~|      *=      -.=     ~|=     {..}
->      +       [:      +.      **      ~&      /=      *.=     ~&=     !&&.=
!       -       :]      -.      ^^      !||     %=      /.=     !||.     .|.
~       *       (:      *.      ||      !&&     :=      %.=     !&&.
^       /       :)      /.      &&      **.     |=      ||=     ||.=
@       %       ::      %.      <<      ###     &=      &&=     &&.=
```

2.9. Комментарии

В любом месте программы могут присутствовать комментарии. Комментарий может занимать любое число строк исходного текста, а начинается с сочетания /* и заканчивается сочетанием */. Кроме того, комментарии могут быть вложенными, без ограничения на уровень вложенности. Приведём пример:

```
t=sin(x)
/* Это комментарий первого уровня вложенности.
/* Это - второго.
/* А это - третьего. */
*/
*/
```

Глава 3. Структура программы

Структура программы на языке Лиса такова:

модуль имя_модуля
{
    описание*
}

Здесь
имя_модуля — идентификатор, являющийся именем данного модуля;
описание — описание типов, переменных, констант и функций.

Глава 4. Описания

Область видимости объекта `x` (здесь под объектом понимается тип, переменная, константа, или функция) текстуально распространяется от точки его описания до конца блока (модуля, тела составного оператора, тела функции), которому принадлежит описание и по отношению к которому объект, таким образом, считается локальным. Из этой области исключаются области видимости объектов с таким же именем, описанных в блоках, вложенных в данный. Правила видимости таковы.

1) Идентификатор может обозначать только один объект в данной области видимости (т.е. никакой идентификатор не может быть объявлен в блоке дважды).

2) На объект можно сослаться только в его области видимости.

3) Описание типа `T`, содержащее ссылки на другой тип $\texttt{T}_1$, может стоять в точках, где $\texttt{T}_1$ еще не известен. Описание типа $\texttt{T}_1$ должно следовать далее в том же блоке, в котором локализован `T`.

4) Заголовок функции может быть приведён до того, как будет дано полное определение.

4.1. Описание типов

Описание типов выглядит так:

тип имя_типа=определение_типа(;имя_типа=определение_типа)*

Здесь

1) имя_типа — идентификатор, являющийся именем определяемого типа;

2) определение_типа — либо простейшее определение типа, либо определение алгебраического типа данных.

Алгебраические типы данных будут подробно рассмотрены в разделе, посвящённом таким типам. Здесь же поясним понятие простейшего определения типа.

Простейшие определения типов есть двух категорий:

1) стандартные типы;

2) простейшие определения типов, задаваемые пользователем.

Стандартные типы можно разделить на четыре вида:

1) логические типы;

2) символьный тип;

3) строковый тип;

4) числовые типы;

5) пустой тип.

В свой черёд, числовые типы могут быть следующих подвидов:

1) целочисленные типы;

2) вещественные типы;

3) комплексные типы.

К простейшим определениям типов, задаваемым пользователем, относятся:

1) имя типа;

2) определение типа–указателя;

3) определение типа указателя на функцию;

4) определение типа–массива.

5) определение типа–множества.

Приведём пример определения типов:

```
тип  A = мал мал цел;
     B = мал цел;
     C = цел;
     D = бол цел
```

Здесь типы `A` и **мал мал цел** — взаимозаменяемы.

Для каждого типа данных можно узнать размер переменной этого типа, для чего перед именем типа или переменной этого типа нужно поставить знак операции `##`.

Для динамических массивов операция `##` даёт размер не самого этого значения, а размер служебной информации. Чтобы узнать размер самого́ значения динамического массива, нужно перед именем переменной поставить знак операции `###`.

4.1.1. Логические типы

Переменная логического типа может принимать только два значения: **истина** или **ложь**. Логический тип выглядит так:

(**бол**|**мал**)* **логик**

Размер переменной типа **логик** зависит от реализации, но не может превышать размера машинного слова. Также имеются логические типы конкретных размеров, а именно, типы **логик8**, **логик16**, **логик32**, **логик64**, переменные которых имеют размер в 1, 2, 4 и 8 байт соответственно.

Над логическими значениями определены следующие операции:

`||`	логическое „или“ (сокращённое вычисление)
`||.`	логическое „или“ (полное вычисление)
`!||`	логическое „не–или“ (сокращённое вычисление)
`!||.`	логическое „не–или“ (полное вычисление)
`&&`	логическое „и“ (сокращённое вычисление)
`&&.`	логическое „и“ (полное вычисление)
`!&&`	логическое „не–и“ (сокращённое вычисление)
`!&&.`	логическое „не–и“ (полное вычисление)
`^^`	логическое „исключающее или“
`!`	логическое „не“
`==`	равно
`!=`	не равно

Все логические типы попарно совместимы между собой. Термин „полное вычисление“ означает, что вычисляются все аргументы операции; а термин „сокращённое вычисление“ — что вычисляется лишь часть аргументов.

Приведём таблицы истинности логических операций.

`x`	`y`	`x||y`	`x&&y`	`x^^y`
ложь	**ложь**	**ложь**	**ложь**	**ложь**
ложь	**истина**	**истина**	**ложь**	**истина**
истина	**ложь**	**истина**	**ложь**	**истина**
истина	**истина**	**истина**	**истина**	**ложь**

x	y	(x!||y)≡x||(!y)	(x!&&y)≡x&&(!y)
ложь	**ложь**	**истина**	**ложь**
ложь	**истина**	**ложь**	**ложь**
истина	**ложь**	**истина**	**истина**
истина	**истина**	**ложь**	**ложь**

x	!x
ложь	**истина**
истина	**ложь**

4.1.2. Символьный тип

Переменная символьного типа может хранить любой символ, доступный в конкретной реализации. Символьный тип выглядит так:

симв

Для символьных данных определены лишь операции отношения и операция присваивания. Ниже приведён список операций отношения:

- `<` меньше
- `>` больше
- `<=` меньше или равно
- `>=` больше или равно
- `==` равно
- `!=` не равно

4.1.3. Строковый тип

Переменная строкового типа хранит строковые значения. Строковый тип выглядит так:

строка

Для строковых данных определены операции отношения, операция присваивания, и операция обращения к символу строки по его индексу. Также определена операция конкатенации (склейки) строк, обозначаемая знаком „+“.

4.1.4. Числовые типы

4.1.4.1. Целочисленные типы

Переменные целочисленных типов хранят целые числа. Целочисленный тип выглядит так:
(**бол*** | **мал***) **бз**? **цел**|**бз8**|**бз16**|**бз32**|**бз64**|**бз128**|**цел8**|**цел16**| **цел32**|**цел64**|**цел128**

Допустимые операции:

+ (бинарный)	целочисленное сложение
+ (унарный)	подтверждение знака
- (бинарный)	целочисленное вычитание
- (унарный)	изменение знака
++	следующее значение
--	предыдущее значение
*	целочисленное умножение
/	целочисленное деление
%	целочисленный остаток от деления
**	целочисленное возведение в степень
|	поразрядное „или“

`~|`	поразрядное „не–или“
`&`	поразрядное „и“
`~&`	поразрядное „не–и“
`~`	поразрядное „не“
`^`	поразрядное „исключающее или“
`<<`	сдвиг влево
`>>`	сдвиг вправо
`<`	меньше
`>`	больше
`<=`	меньше или равно
`>=`	больше или равно
`==`	равно
`!=`	не равно

4.1.4.2. Вещественные типы

Переменные вещественных типов предназначены для хранения вещественных чисел. Вещественный тип выглядит так:

(**бол*** |**мал***)**вещ**|**вещ32**|**вещ64**|**вещ80**|**вещ128**

Допустимые операции:

`+.`	сложение
`+` (унарный)	подтверждение знака
`-.`	вычитание
`-` (унарный)	изменение знака
`*.`	умножение
`/.`	деление
`%.`	вещественный остаток от деления
`**.`	вещественное возведение в степень
`<`	меньше
`>`	больше
`<=`	меньше или равно
`>=`	больше или равно
`==`	равно
`!=`	не равно

4.1.4.3. Комплексные типы

Переменные комплексных типов предназначены для хранения комплексных типов. Комплексный тип выглядит так:

(**бол*** |**мал***)**компл**|**компл32**|**компл64**|**компл80**|**компл128**

Допустимые операции:

`+.`	сложение
`+` (унарный)	подтверждение знака
`-.`	вычитание
`-` (унарный)	изменение знака
`*.`	умножение
`/.`	деление
`==`	равно
`!=`	не равно

4.1.5. Пустой тип

Пустой тип — это тип **ничего**. Тип **ничего** может быть либо базовым типом указателя, либо типом значения функции. Ни в каких других целях тип **ничего** применяться не может. При этом **##ничего**= 0, т.е. размер типа **ничего** равен нулю.

4.1.6. Кортежи

Кортеж — это упорядоченный набор конечного числа элементов, вообще говоря, разных типов. Тип–кортеж выглядит так:

(:(тип_элемента(,тип_элемента)*)?:)

Здесь тип_элемента — тип соответствующего элемента кортежа. Этот тип может быть либо именем типа, либо указателем, либо типом указателя на функцию, либо встроенным типом, либо кортежем.

Если для каждого элемента кортежа определена одна и та же операция отношения, то эта операция определена и для всего кортежа.

Кроме того, если x — значение–кортеж, то можно получить значения отдельных элементов этого кортежа. А именно, для получения значения элемента с номером i (элементы кортежа нумеруются слева направо, и нумерация начинается с нуля), нужно написать $x\#i$.

4.1.7. Указатели

Указатели содержат адреса ячеек памяти. Тип–указатель определяется так:

@базовый_тип_указателя

Указателю можно присвоить константу **ничего**. В этом случае указатель перестаёт указывать на какую бы то ни было ячейку памяти. Указатель можно разыменовывать, то есть получать значение переменной, на которую он указывает. Для разыменования указателя нужно после имени указателя поставить знак @. Разыменовывать можно все указатели, кроме указателей типа **@ничего**. Тип переменной, на которую указывает указатель, называется базовым типом указателя.

Указатели можно сравнивать на равенство и неравенство.

Указателю типа **@ничего** можно присваивать значение указателя любого типа.

Пример 4.1.1.

```
перем x : цел;
      y : @цел
      ...
      x := y@ + 1;
      ...
```

Если нужно получить адрес какой–либо переменной, то перед именем этой переменной нужно записать знак @. Результатом данной операции является указатель, базовый тип которого — тип переменной. При этом для динамических массив таким образом будет получен лишь адрес области служебных данных. Чтобы получить указатель на начало той области, в которой хранятся данные динамического массива, нужно перед именем переменной, тип которой — динамический массив, поставить знак @@.

4.1.8. Ссылки

Тип–ссылка выглядит так:

(**ссылка**|**конст ссылка**)базовый_тип_ссылки

4.1.9. Типы указателей на функции

Переменные таких типов предназначены для хранения указателей на функции. Тип указателя на функцию выглядит так:

чистая?**фун** сигнатура

Здесь сигнатураопределяется следующим образом:
сигнатура→((группа_параметров(;группа_параметров)*)?):тип_значения
группа_параметров→имя_параметра(,имя_параметра)*
имя_параметра→идентификатор
Ключевое слово **чистая** означает, что у функции нет побочных эффектов.

4.1.10. Массивы

Тип–массив имеет следующий вид:

массив[выражение?(,выражение?)*] тип_элемента_массива

Каждое из выражений указывает, сколько значений может принимать соответствующий индекс массива. Если какое–либо из выражений опущено, то по этому измерению массив считается динамическим. Каждое выражение должно быть таким, чтобы его можно было вычислить на этапе компиляции. Наименьшее значение каждого индекса равно нулю, а массивы хранятся по строкам.

При этом записи

$$\textbf{массив}[\mathtt{N}_0,\ldots,\mathtt{N}_{m-1}]\ \textbf{массив}[\mathtt{N}_m,\ldots,\mathtt{N}_{m+p-1}]\ \mathtt{T}$$

и

$$\textbf{массив}[\mathtt{N}_0,\ldots,\mathtt{N}_{m-1},\mathtt{N}_m,\ldots,\mathtt{N}_{m+p-1}]\ \mathtt{T}$$

считаются эквивалентными.

Далее, если тип T определён как

$$\textbf{массив}[\mathtt{N}_m,\ldots,\mathtt{N}_{m+p-1}]\ \mathtt{T}'$$

то запись

$$\textbf{массив}[\mathtt{N}_0,\ldots,\mathtt{N}_{m-1}]\ \mathtt{T}$$

считается эквивалентной записи

$$\textbf{массив}[\mathtt{N}_0,\ldots,\mathtt{N}_{m-1},\mathtt{N}_m,\ldots,\mathtt{N}_{m+p-1}]\ \mathtt{T}'$$

Кроме того, любой тип вида

$$\textbf{массив}[\mathtt{N}_0,\ldots,\mathtt{N}_{m-1}]\ \mathtt{T}$$

где тип T эквивалентен типу **ничего**, сам эквивалентен типу **ничего**.

Все эти преобразования производятся на этапе компиляции.

Пример 4.1.2. *Пусть сделаны определения*

```
конст N : цел = 128
тип   float = мал вещ;
      T = массив[N] float
```

Тогда следующие записи эквивалентны:

1) **массив**[N, N]float
2) **массив**[N] **массив**[N] float
3) **массив**[N] T

Для обращения к элементу массива надо после имени массива в квадратных скобках перечислить индексы нужного элемента.

Пример 4.1.3. *Пусть сделаны определения*

```
конст N : цел = 128;
      M : цел = 256

тип   float = мал вещ;
      T1 = массив[M] float;
      T2 = массив[N] T1
```

перем A : T1;
B : T2

Тогда к элементу массива `A` *с индексом 200 нужно обращаться как* `A[200]`*, а к имеющему индекс 107 элементу массива* `B` *—* `B[107]`*. Поскольку, в силу сделанных определений, элемент* `B[107]` *сам является массивом, то для обращения к имеющему индекс 91 элементу массива* `B[107]` *нужно писать* `B[107][91]`*. Последняя запись эквивалентна записи* `B[107, 91]`*. Аналогичные правила действуют и для массивов большей размерности.*

Тип элемента массива называется базовым типом массива.

Для массивов определена инфиксная бинарная операция `#`, первым (левым) операндом которой служит имя массива, а вторым (правым) — номер индекса массива, считая слева. Самый левый индекс имеет номер ноль. В результате вычисления данной операции будет получено количество возможных значений указанного вторым операндом индекса. Так происходит, если второй аргумент неотрицателен и меньше количества индексов (с учётом преобразований этапа компиляции). Если же второй аргумент операции `#` либо отрицателен, либо не меньше количества индексов массива, то результат будет равен нулю.

Пример 4.1.4. *Пусть сделаны определения*
перем A : **массив**[16]**вещ**;
B : **массив**[9, 11]**вещ**;
C : **массив**[17, 8, 19]**вещ**

Тогда `A#`$0 = 16$, `A#`$(-3) = 0$, `B#`$0 = 9$, `B#`$(-5) = 0$, `B#`$1 = 11$, `B#`$2 = 0$, `B#`$1000 = 0$, `A#`$1 = 0$, `B[3]#`$0 = 11$, `C#`$0 = 17$, `C#`$1 = 8$, `C#`$2 = 19$, `C[5]#`$0 = 8$, `C[5]#`$1 = 19$, `C[5,4]#`$0 = 19$.

4.1.11. Перечислимые типы

Перечислимым называется тип, в определении которого указаны все его возможные значения, являющиеся идентификаторами. Среди этих идентификаторов не должно быть повторяющихся. Определение перечислимого типа выглядит так:

имя_типа = перечисление имя_перечисления{имя_значения(,имя_значения)* }

Здесь имя_значения — идентификатор, являющийся именем значения перечислимого типа.

Ниже приведены примеры определения перечислимых типов.

тип светофор = **перечисление** сф{красный, жёлтый, зелёный};
день_недели = **перечисление** дн{понедельник, вторник, среда, четверг,
пятница, суббота, воскресенье};
месяц_года = **перечисление** мг{январь, февраль, март, апрель, май, июнь, июль,
август, сентябрь, октябрь, ноябрь, декабрь}

Для значений перечислимого типа допустимы следующие операции:

`++`	следующее значение
`--`	предыдущее значение
`++<`	следующее значение с заворачиванием
`--<`	предыдущее значение с заворачиванием
`<`	меньше
`>`	больше
`<=`	меньше или равно
`>=`	больше или равно
`=`	равно
`!=`	не равно

Поясним смысл этих операций.

Пусть сделано определение вида

тип T = **перечисление**V{z_0, z_1,..., z_{m-1}}

где z_i, $i = \overline{0, m-1}$, — какие–то попарно различные идентификаторы. Пусть, кроме того, x и y —

значения типа T. Тогда

$$\texttt{++x} = \begin{cases} \texttt{z}_{i+1}, \text{ если } \texttt{x=z}_i,\ i = \overline{0, m-2}; \\ \text{не определено, если } \texttt{x=z}_{m-1}; \end{cases} \qquad \texttt{++<x} = \begin{cases} \texttt{z}_{i+1}, \text{ если } \texttt{x=z}_i,\ i = \overline{0, m-2}; \\ \texttt{z}_0, \text{ если } \texttt{x=z}_{m-1}; \end{cases}$$

$$\texttt{--x} = \begin{cases} \texttt{z}_{i-1}, \text{ если } \texttt{x=z}_i,\ i = \overline{1, m-1}; \\ \text{не определено, если } \texttt{x=z}_0; \end{cases} \qquad \texttt{-->x} = \begin{cases} \texttt{z}_{i-1}, \text{ если } \texttt{x=z}_i,\ i = \overline{1, m-1}; \\ \texttt{z}_{m-1}, \text{ если } \texttt{x=z}_0; \end{cases}$$

$\texttt{(x < y)} \stackrel{def}{=} (\texttt{код(x)} < \texttt{код(y)})$, $\texttt{(x <= y)} \stackrel{def}{=} (\texttt{код(x)} \leqslant \texttt{код(y)})$, $\texttt{(x > y)} \stackrel{def}{=} (\texttt{код(x)} > \texttt{код(y)})$,

$\texttt{(x >= y)} \stackrel{def}{=} (\texttt{код(x)} \geqslant \texttt{код(y)})$, $\texttt{(x = y)} \stackrel{def}{=} (\texttt{код(x)} = \texttt{код(y)})$, $\texttt{(x != y)} \stackrel{def}{=} (\texttt{код(x)} \neq \texttt{код(y)})$.

При этом по определению $\texttt{код(x)} = i$, если $\texttt{x} = \texttt{z}_i$ $(i = \overline{0, m-1})$.

4.1.12. Множества

Множеством назовём совокупность объектов, которое может состоять из произвольного числа элементов одного типа. Тип-множество имеет следующий вид:

множество *имя_типа*

Здесь имя_типа – идентификатор, являющийся именем перечислимого типа.

Каждый элемент множества должен быть таким, чтобы его можно было вычислить на этапе компиляции. В множестве не должно быть повторяющихся элементов.

Число элементов множества не может превышать число значений, входящих в простейшее определение типа.

Присваивание значений. Объектами множественного типа являются конкретные множества. Обращаться к ним можно с помощью переменных, вводимых обычным путём.

Пример 4.1.5.

тип месяц = **перечисление** мес(янв, февр, март, апр, май, июнь, июль, авг, сент, окт, нояб, дек);
A = **множество** месяц;
перем m : A;

тип N = **множество** цел;
перем i : N;
i := (1,2,5,7);

В данном случае значение переменной m может быть произвольная совокупность названий месяца; значение переменной i – определённый набор цифр из множества целых чисел.

Множество задаётся в виде списка элементов, заключённого в круглые скобки. В скобках может не быть элементов, может стоять только один элемент или может присутствовать несколько элементов, разделённых запятыми. В качестве элемента может выступать константа, переменная или выражение, значения которых принадлежат определению типа множества, а также пара элементов, разделённых двумя точками, т.е. интервал значений.

Следовательно, можно использовать инструкции присваивания следующих видов:

m := (); (* пустое множество *)
m := (янв); (* множество включает только один элемент *)
m := (янв .. март, авг, нояб .. дек); (* в множество входят элементы янв, февр, март, авг, нояб, дек *)

Операции над множествами. Средства, предназначенные для создания множеств и присваивания им значений, не позволяют в полной мере использовать возможности данных множественного типа. В связи с этим введены такие операции:

1) **Объединение**
   В объединение множеств L + R входят элементы, которые являются членами хотя бы одного из множеств L или R.

2) **Пересечение**
   В пересечение множеств L * R входят элементы, принадлежащие и множеству L, и множеству R.

3) **Разность**
   В разность множеств L – R входят элементы, которые являются членами множества L и не являются членами множества R.

Пример 4.1.6. *Пусть сделаны определения*
тип C = **множество**(анна, елена, соня, геля, лиза, мария, света, саша);
перем A, B : C;

и две конструкции присваивания:
A := (анна, соня, мария .. саша);
B := (анна, лиза, мария, света);

В результате выполнения двум множествам A и B (в множество A включены имена спортивных девушек, а в множество B – имена девушек, которые занимаются рисованием) присваиваются конкретные значения.

Тогда результатом A + B является множество имён тех девушек, которые или занимаются спортом, или хороши в искусстве, или и спортивны, и рисуют. В это множество войдут анна, соня, лиза, мария, света, саша.

Результатом A * B является множество, в которое входят имена спортивных и творческих девушек, т.е. анна, мария, света.

Наконец, результатом A – B является множество имён девушек, которые спортивны, но не занимаются рисованием. Это соня и саша.

Оператор **в** позволяет определить, принадлежит элемент множеству или нет. Первым операндом, расположенным слева от слова **в**, является значением простейшего определения типа, а вторым операндом, стоящим справа, имя множества. Результатом являются значения типа boolean: true – если значение является элементом множества; false – если не является. Следовательно,

лиза **в** B
даёт в результате true, а
геля **в** (A + B)
даёт false

При работе с множествами можно использовать операторы сравнения. Операторы = и < > позволяют проверить, равны два множества или нет. Операторы <= и >= называются операторами включения. С их помощью легко определить, является ли одно множество подмножеством другого. Например, отношение
(анна, света) <= (A + B)
истинно, поскольку все элементы, стоящие слева, содержатся в множестве, расположенном справа.

Расположим операторы, предназначенные для работы с множествами, в порядке убывания приоритета:

1) *****

2) **+, –**

3) **в, =, < >, <=, >=**

В группы объединены операторы равного приоритета. Последовательность выполнения операторов одного приоритета определяется порядком их появления в выражении. Для изменения порядка выполнения операторов используются круглые скобки.

4.1.13. Структуры

Структура — это упорядоченный набор компонент, вообще говоря, различных типов. Каждая такая компонента называется полем структуры. Синтаксис описания типа–структуры:

тип_структура = **структ** имя_структуры {группа_полей(;группа_полей)* }

группа_полей→имя_поля(,имя_поля)* :определение_типа

имя_поля→идентификатор

Пример 4.1.7. *Пусть сделаны определения*

```
тип дата =  структ дата{
                день: цел8;
                месяц: цел8;
                месяц: цел16
    }
перем x : дата
```

Тогда для обращения к полю `день` *переменной* `x` *нужно записать* `x.день`.

4.1.14. Алгебраические типы данных

Более общим способом определения типа, по сравнению с определением типа–структуры и типа–перечисления, является определение алгебраического типа данных. Алгебраический тип данных определяется так:

опр_алгебр_типа→имя_типа = компонента (.|.компонента)*

компонентаопр_структуры|опр_перечисления

опр_структуры→**структ** имя_структуры {тело_структуры?}

тело_структуры→группа_полей(;группа_полей)*

группа_полей→имя_поля(,имя_поля)* :определение_типа

имя_поля→идентификатор

имя_структуры→идентификатор

имя_типа→идентификатор

опр_перечисления→**перечисление** имя_перечисления{имя_значения(,имя_значения)* }

имя_значения→идентификатор

Здесь разделитель .|. означает „или“.

4.1.15. Эквивалентность и совместимость типов

Выше использовалось понятие эквивалентности типов. Определим это понятие. Но сначала дадим определение *правильного типа–массива*.

Тип–массив T назовём *правильным типом–массивом*, если T имеет вид **массив**[N_0,...,N_{m-1}]T′, где T′ не является массивом, а N_0,...,N_{m-1} — неотрицательные целочисленные константы. Для каждой из этих констант положительное значение имеет смысл количества значений соответствующего индекса массива; а нулевое означает, что по этому измерению массив является динамическим. Если тип T′ — это тип **ничего**, то считаем, что тип T эквивалентен типу **ничего**.

На этапе компиляции каждый тип преобразуется к виду {@}T, где T — что–либо из следующего списка:

1) символьный тип;
2) логический тип;
3) целочисленный тип;
4) вещественный тип;
5) комплексный тип;
6) пустой тип;
7) алгебраический тип;

8) тип указателя на функцию;

9) правильный тип–массив.

Такой вид типа назовём *каноническим*. Будем считать, что все компоненты тех компонент алгебраических типов, которые являются структурами, приведены к каноническому виду.

Количество символов @ в каноническом виде типа T будем называть *ссылочным порядком типа T*, и обозначать сп(T). Эквивалентность типов T_1 и T_2 будем обозначать так: $T_1 \sim T_2$.

Наконец, дадим определение эквивалентности типов.

Определение 4.1.1.

1) *Для любого типа T справедливо соотношение $T \sim T$.*

2) *Если $T_1 \sim T_2$, то $T_2 \sim T_1$.*

3) *Если $T_1 \sim T_2$ и $T_2 \sim T_3$, то $T_1 \sim T_3$.*

4) *Разные стандартные типы не эквивалентны.*

5) *Если сделано определение вида* **тип** T_1 = T_2, *то $T_1 \sim T_2$.*

6) *Если сп$(T_1) \neq$ сп(T_2), то $T_1 \not\sim T_2$.*

7) *Пусть даны два типа, T_1 и T_2, у которых сп$(T_1) =$ сп$(T_2) > 0$. Пусть, кроме того, $\{@\}T'_1$ — канонический вид типа T_1, а $\{@\}T'_2$ — канонический вид типа T_2. Тогда $T_1 \sim T_2$ в том и только в том случае, когда $T'_1 \sim T'_2$.*

8) *Два правильных типа–массива эквивалентны тогда и только тогда, когда*

   а) *у массивов — одинаковое количество индексов;*

   б) *количество значений соответствующих индексов совпадает;*

   в) *базовые типы эквивалентны.*

9) *Два типа указателей на функции эквивалентны только при одновременном выполнении следующих условий:*

   а) *количество формальных параметров совпадает;*

   б) *соответствующие параметры имеют эквивалентные типы;*

   в) *типы возвращаемых значений эквивалентны;*

   г) *либо оба типа чистые (т.е. с атрибутом* **чистая***), либо оба типа — грязные (т.е. без указанного атрибута).*

10) *Во всех не упомянутых здесь случаях типы неэквивалентны.*

При использовании в программе эквивалентные типы взаимозаменяемы.

При вызове функции необходимо, чтобы типы формальных и фактических параметров были определённым образом согласованы. Можно было бы отождествить такую согласованность с эквивалентностью типов формальных и фактических параметров, но это было бы слишком жёстким требованием. Таким образом, нам необходимо понятие согласованности типов, отличающееся от понятия эквивалентности типов. Нужную нам согласованность назовём *совместимостью* типов. Тот факт, что тип T_1 совместим с типом T_2, будем обозначать так: $T_1 \rhd T_2$.

Дадим определение совместимости типов.

Определение 4.1.2.

1) *Если $T_1 \sim T_2$, то $T_1 \rhd T_2$ и $T_2 \rhd T_1$.*

2) *Каждый из алгебраических типов совместим только сам с собою.*

3) *Каждый из символьных типов совместим только сам с собою.*

4) *Все логические типы попарно совместимы между собой.*

5) *Если* T_1 *и* T_2 *— целочисленные типы и диапазон значений типа* T_1 *содержится в диапазоне значений типа* T_2*, то* $T_1 \rhd T_2$.

6) *Если* T_1 *и* T_2 *— вещественные типы и точность типа* T_1 *не превышает точности типа* T_2*, то* $T_1 \rhd T_2$.

7) *Если* T_1 *и* T_2 *— комплексные типы и точность компонент типа* T_1 *не превышает точности компонент типа* T_2*, то* $T_1 \rhd T_2$.

8) *Если* T_1 *— целочисленный тип, а* T_2 *— вещественный или комплексный тип, то* $T_1 \rhd T_2$.

9) *Если* T_1 *— вещественный тип, а* T_2 *— комплексный тип, причём точность типа* T_1 *не превышает точности компонент типа* T_2*, то* $T_1 \rhd T_2$.

10) *Пусть* T_1 *и* T_2 *— типы–массивы. Обозначим через* B_1 *и* B_2 *базовые типы типов* T_1 *и* T_2 *соответственно; через* d_1 *и* d_2 *— количество индексов у* T_1 *и* T_2 *соответственно; наконец, через* N_1^j, $j = \overline{0, d_1 - 1}$, *обозначим количество значений индекса с номером* j *у массива* T_1*, а через* N_2^j, $j = \overline{0, d_2 - 1}$, *— количество значений индекса с номером* j *у массива* T_2*. Тогда* $T_1 \rhd T_2$ *только при одновременном выполнении следующих условий:*

   а) $d_1 = d_2$;

   б) $B_1 \rhd B_2$;

   в) *если* $N_2^j = 0$*, то* N_1^j *может быть любым; в противном случае должно выполняться равенство* $N_1^j = N_2^j$ $(j = \overline{0, d_1 - 1})$.

11) *Во всех не упомянутых здесь случаях типы несовместимы.*

4.2. Описание переменных

Синтаксис описания переменных:
описание_переменных→**перем** группа_переменных:определение_типа(= выражение)?
(;группа_переменных:определение_типа(= выражение)?)*
группа_переменных→имя_переменной(,имя_переменной)*
имя_переменной→идентификатор

4.3. Описание констант

Синтаксис описания констант:
описание_констант→имя_константы : тип_константы = значение_константы
(;имя_константы : тип_константы = значение_константы)*
имя_константы→идентификатор
значение_константы→выражение

4.4. Описание функций

Описание функции имеет следующую структуру:
описание_функции→(**глав** |**чистая**)?**фун** имя_функции сигнатура(;|реализация)
реализация→{(описание|операторы)* }
имя_функции→идентификатор

Необязательное ключевое слово **глав** означает, что выполнение модуля начинается с этой функции. Функций с атрибутом **глав** в модуле может быть не более одной.

Необязательное ключевое слово **чистая** означает, что функция не имеет побочных эффектов.

Глава 5. Выражения

Синтаксис выражений выглядит так:

выражение→выражение$_0$(операция_присваивания выражение)?

операция_присваивания→`=` |`:=` |`||=` |`!||=` |`|||.=` |`!||.=` |`&&=` |`&&.=` |`!&&=` |`!&&.=` |`^^=` |`|=` |`&=` |
`~|=` |`~&=` |`^=` |`<<=` |`>>=` |`+=` |`-=` |`*=` |`/=` |`%=` |`**=` |`+.=` |`-.=` |`*.=` |`/.=` |
`%.=` |`**.=`

выражение$_0$→выражение$_1$((`?` |`?.`)выражение$_1$: выражение$_1$)?

выражение$_1$→выражение$_2$((`||` |`||.` |`!||` |`!||.` |`^^`)выражение$_2$)*

выражение$_2$→выражение$_3$((`&&` |`&&.` |`!&&` |`!&&.`)выражение$_3$)*

выражение$_3$→(`!`)* выражение$_4$

выражение$_4$→выражение$_5$((`<` |`>` |`<=` |`>=` |`==` |`!=`)выражение$_5$)*

выражение$_5$→выражение$_6$((`|` |`~|` |`^`)выражение$_6$)*

выражение$_6$→выражение$_7$((`&` |`~&` |`<<` |`>>`)выражение$_7$)*

выражение$_7$→(`~`)* выражение$_8$

выражение$_8$→выражение$_9$((`+` |`+.` |`-` |`-.`)выражение$_9$)*

выражение$_9$→выражение$_{10}$((`*` |`*.` |`/` |`/.` |`%` |`%.`)выражение$_{10}$)*

выражение$_{10}$→выражение$_{11}$((`**` |`**.`)выражение$_{10}$)?

выражение$_{11}$→выражение$_{12}$(`#`выражение$_{12}$)?

выражение$_{12}$→(`++`|`--` |`++<` |`--<`)* выражение$_{13}$

выражение$_{13}$→(`+` |`-`)выражение$_{14}$

выражение$_{14}$→(`@` |`@@` |`##` |`###`)?выражение$_{15}$

выражение$_{15}$→выделение_памяти |освобождение_памяти |литерал |
имя |(выражение) |составное_значение

выделение_памяти→**выдел**(`[:`выражение(`,`выражение)* `:]`)?имя

освобождение_памяти→**очищ**(`[::]`)?имя

литерал→символьный |строковый |целое |вещественное |комплексное |**истина** |**ложь** |**ничего**

имя→(идентификатор`::`)* идентификатор(`.`идентификатор |`@` |`(`(выражение(`,`выражение)*)?`)` |
`[`(выражение(`,`выражение)*)?`]`)*

составное_значение→значение_структуры |значение_массива

значение_структуры→идентификатор`{`(имя_поля`<-`выражение(`,`имя_поля`<-`выражение)*)?`}`

значение_массива→**массив**`[:`выражение?(`,`выражение?)* `:]{`выражение(`,`выражение)* `}`

Глава 6. Операторы

6.1. Операторы присваивания

Синтаксис оператора присваивания:

имя операция_присваивания выражение

Все эти операторы можно разделить на три группы: простой оператор присваивания (=), оператор копирования (:=) и все прочие операторы присваивания. Отличие оператора копирования от оператора присваивания состоит в поведении для динамических массивов: в этом случае оператор присваивания копирует ссылки на значения (точнее, служебные сведения, в которые входят эти ссылки), а оператор копирования копирует сами значения.

6.2. Условный оператор

Синтаксис условного оператора:
условный_оператор→**если** условие **то**{(описание |операторы)* }
(**иначеесли** условие **то**{(описание |операторы)* })*
иначе{(описание |операторы)* }
Здесь условие — это логическое выражение, а **иначеесли** условие **то**{(описание |операторы)* } является сокращённой формой для **иначе**{**если** условие **то**{(описание |операторы)* }}.

6.3. Оператор выбора

Синтаксис оператора выбора:
оператор_выбора→**выбор** выражение {
список_значений_для_ветви : {ветвь}
(список_значений_для_ветви : {ветвь})*
(**иначе**{ветвь_иначе})?
}
список_значений_для_ветви→значение_для_ветви(,значение_для_ветви)*
значение_для_ветви→выражение(..выражение)?
ветвь→(описание |операторы)*
ветвь_иначе→ветвь
Оператор выбора определяет выбор и выполнение операторов на основе значения выражения S, которое должно быть выражением целочисленного, перечислимого, или символьного типа.

Оператор выполняется так. Сначала вычисляется выражение, идущее сразу после ключевого слова **выбор**. Затем выполняется та ветвь (т.е. последовательность операторов), соответствующий которой список значений содержит значение выражения S. При этом можно указывать диапазоны значений, а именно, следующим образом: `A..B`, где `A` — минимальное значение в диапазоне, `B` — максимальное значение в диапазоне. Например, `1..2000`, `-7..7`. Значения в списках значений должны быть константами, которые можно вычислить на этапе компиляции, и ни одно значение не должно употребляться более одного раза. Если нужного значения нет в списке значений ни для какой ветви, то выполняются операторы ветви ветвь_иначе, если ключевое слово **иначе** присутствует. Если же ключевого слова **иначе** нет, то выполнение оператора выбора завершается.

6.4. Оператор разбора значения алгебраического типа

Синтаксис оператора разбора:
оператор_разбора→**разбор** выражение {
метка_разбора->{ветвь}

(метка_разбора->{ветвь})*
(**иначе**{ветвь_иначе})?
}
метка_разбора→имя_компоненты_алгебраического_типа{..}
имя_компоненты_алгебраического_типа→(идентификатор::)* идентификатор

6.5. Операторы цикла

Операторы цикла организуют выполнение повторяющихся действий. Всего в языке есть четыре типа операторов цикла: оператор цикла с предусловием (оператор **„пока“**), оператор цикла с постусловием (оператор **„повт. . . пока“**), оператор **„бескон повт“**, оператор **„для“**. Опишем каждый из этих операторов.

6.5.1. Оператор цикла с предусловием

Оператор цикла с предусловием выглядит так:

(+|имя_цикла:)?**пока**(условие){(описание |операторы)* }

Здесь условие — это логическое выражение, а имя_цикла — идентификатор, являющийся именем цикла. Данный идентификатор можно использовать только в операторе выхода из цикла. Оператор **„пока“** выполняет тело цикла, пока логическое выражение условие остаётся истинным. Истинность этого логического выражения проверяется перед каждым выполнением тела цикла.

6.5.2. Оператор цикла с постусловием

Оператор цикла с постусловием выглядит так:

(+|имя_цикла:)?**повт**{(описание |операторы)* }**пока**(условие)

Здесь условие — это логическое выражение, а *имя_цикла* — идентификатор, являющийся именем цикла. Данный идентификатор можно использовать только в операторе выхода из цикла. Оператор цикла **„повт. . . пока“** выполняет тело цикла, пока логическое выражение условие остаётся истинным. Истинность этого логического выражения проверяется после каждого выполнения тела цикла.

6.5.3. Оператор цикла „бескон повт“

Оператор **„бескон. . . повт“** выглядит так:

(+|имя_цикла:)?**бескон повт**{(описание |операторы)* }

Здесь *имя_цикла* — идентификатор, являющийся именем цикла. Данный идентификатор можно использовать только в операторе выхода из цикла. Оператор **„бескон повт“** выполняется до тех пор, пока из него не будет совершён явный выход — либо с помощью оператора выхода из цикла, либо с помощью оператора возврата из подпрограммы.

6.5.4. Оператор цикла „для“

Оператор цикла **„для“** выглядит так:

(+|имя_цикла:)?**для** *v* = `нач_знач`, `кон_знач`(, `шаг`)?{(описание |операторы)* }

Здесь *v* — идентификатор, являющийся именем переменной цикла; `нач_знач` — начальное значение переменной цикла; `кон_знач` — конечное значение переменной цикла; `шаг` — шаг цикла. По умолчанию шаг равен единице. Величины `нач_знач`, `кон_знач` и `шаг` являются выражениями, вычисляемыми до начала цикла. Переменная цикла должна быть символьного, целочисленного или перечислимого типа. Выражения `нач_знач` и `кон_знач` должны иметь тип, совместимый с типом переменной *v*, а выражение `шаг` должно быть целочисленного типа. Менять в теле цикла значение переменной цикла нельзя.

Смысл оператора цикла **„для“**:

```
t1 := нач_знач;
t2 := кон_знач;
t3 := шаг;
если(t3 > 0)то
{
      v := t1;
      пока(v <= t2)
      {
            (описание |операторы)*
            увелич(v, t3)
      }
}иначеесли(t3 < 0)то
{
      v := t1;
      пока(v >= t2)
      {
            (описание |операторы)*
            увелич(v, t3)
      }
}иначе
{
      v := t1;
      (описание |операторы)*
      пока(t1 != t2)
      {
            (описание |операторы)*
      }
}
}
```

Часть I

Стандартная библиотека