ТУБ №13:
Метаязыки. РБНФ и синтаксические диаграммы

Языки программирования не зря называют языками. В них всё точно так же, как и в русском, английском, французском, турецком и даже китайском. Ну, в каком-то смысле… Каким бы ни был язык программирования, его определяют те же три элемента, что и у естественных языков: алфавит, синтаксис и семантика.

> Алфавит языка — это множество символов, которые могут использоваться для записи программ на этом языке.
>
> Синтаксис языка — это набор правил, по которым символы и группы символов языка должны записываться для того, чтобы образовывать более сложные конструкции, фразы языка.
>
> Семантика языка — это смысл, значение тех фраз, которые записываются на данном языке.

Начнём мы, конечно же, с алфавита. Если всю жизнь пользоваться одним-двумя естественным языками, может казаться, что алфавит не так уж и важен. Ну, в самом деле, kakaya raznica, kakim alfavitom polzovatsya? Ваше мнение сильно поменяется, если однажды вечером Ваши родители напишут:

— 你好吗？

И поскольку хорошее воспитание не позволит Вам спросить «Это что ещё за котопёс?!», тогда-то точно придётся мучительно разбираться, откуда там взялись лошадь (马) и ребёнок (子).

С языками программирования всё примерно так же. Есть, например, старый-добрый Basic:

```
PRINT "Hello, Vasya!"
```

И вроде бы всё понятно: напечатает куда-то текст «Приветствую, Василий!» А потом открываете Вы программу на другом языке, а там…

```
++++++++++[>+++++++>++++++++++>++++>+++>+++++++++>++++++++++<<<<<<-
]>++.>+.+++++++..+++.>++++.>++.>----.>---.<<<<++++.++++++.>>>>.<<+.
```

Наверное, именно так для китайцев выглядел английский язык, когда они впервые его увидели. А ведь в этом примере всего-то алфавит языка сократился до 7 символов (ещё один символ, входящий в алфавит языка, здесь просто не был использован).

Как правило, в алфавит языка программирования не включаются символы, которые не участвуют в формировании команд языка. Например, во многих языках программирования буквы русского алфавита могут использоваться только для записи текстовой информации, подлежащей обработке, но не для описания конкретных действий самой программы. Другими словами, хотя в то же самом Basic’е и можно написать

```
PRINT "Превед, медвед!"
```

всё равно считается, что алфавит языка ограничивается латиницей, пробелами, цифрами и некоторыми знаками препинания.

Даже из приведённых здесь примеров можно понять, что не любая комбинация символов алфавита языка будет считаться правильной. В естественных языках тоже так: дрове ыб и жномо гадостьдая, о чём речь, но как-то это уже совсем не русский язык. Да и правильно записанные слова тоже должны идти в определённом порядке и правильно разделяться знаками препинания, чтобы не случилось неловких ситуаций вроде слогана

> Каждому пассажиру по мягкому месту!

объявления о знакомстве

> Парень. Ищу добрую, ласковую, нежную, женственную девушку, такую же, как и я.

или совета

> Сапоги надо чистить с вечера, чтобы утром надевать на свежую голову.

Конечно, в большинстве случаев как в естественных языках, так и в языках программирования нарушение правил построения фраз (т.е. синтаксиса) достаточно очевидно. Тем не менее, в языках программирования подобные неловкости тоже возможны, но заканчиваются чаще всего не так весело, особенно если программа управляет самолётом или атомной электростанцией.

Разумеется, при создании новых языков программирования грамотные разработчики стремятся сделать подобные ситуации с неоднозначной трактовкой фраз языка невозможными, но, к сожалению, не каждый, кто создаёт свой язык программирования, достаточно грамотен, поэтому-то выбор языка программирования — это очень важное и ответственное решение.

Синтаксис языка программирования можно объяснить по-разному. Можно, например, приводить десятки, сотни и тысячи примеров программ. Но поможет ли это понять всё правила языка? Вот Вам, например, 10 фраз на турецком языке:

> 1) Hayatınızın tadını çıkarın.
> 2) O zaman da hayatınız herhangi bir filmden daha güzel olur.
> 3) En küçük inanç tohumu, mutluluğun en büyük meyvesinden daha iyidir.
> 4) Aşk fark edilmeden gelir, ama hayata bir kasırga, sonra bir sükunet getirir.
> 5) Bugünün işini yarına bırakma.
> 6) Sakla samanı, gelir zamanı.
> 7) Bozuk saat bile günde iki defa doğruyu gösterir.
> 8) Boşa harcadığın bir dakika, ömründen çaldığın bin dakika.
> 9) Demir tavında dövülür.
> 10) Söylediklerinden hiç bir şey anlamadım.

Чувствуете? Прислушайтесь к ощущениям! Вы ведь уже готовы идти и дискутировать с носителем языка о проблемах освоения космического пространства, разве нет?

Проблема в том, что примеры не могут охватить весь язык, и даже если Вам начать рассказывать, что означает каждое слово и как они между собой связываются в предложения, Вы всего лишь узнаете пару десятков слов и пять-шесть способов собрать их в предложения. Кроме того, почти наверняка какие-то объяснения будут поняты неправильно, слишком узко.

Метаязыки

Чтобы избежать всех этих проблем, а также чтобы не отвечать на миллионы вопросов «А как записать вот такую команду?», разработчики языков программирования придумали метаязыки.

> Метаязык — система обозначений, используемая для записи правил синтаксиса языка.

Можно сказать, что метаязык — это язык, который используется для описания других языков. Два метаязыка, получивших наиболее распространение (иногда с небольшими изменениями), — это расширенная форма Бэкуса-Наура и синтаксические диаграммы.

Сейчас мы перечислим основные понятия, относящиеся к метаязыкам, — и будет вообще непонятно, о чём идёт речь. Обязательно дождитесь примеров, не переключайтесь.

> Метапеременная — это элемент метаязыка, задающий произвольный фрагмент программы.
>
> Метаконстанта — это элемент метаязыка, задающий фрагмент программы, который в описываемом языке должен записываться строго определённым образом.
>
> Метасимвол — это элемент метаязыка, задающий связь между метаконстантами и метапеременными.
>
> Синтаксическая единица — это отдельное правило, описывающее состав и порядок следования элементов при записи конструкций языка.

Расширенная форма Бэкуса-Наура

Расширенная форма Бэкуса-Наура (РБНФ) была предложена швейцарским учёным, одним из известнейших специалистов в области разработки языков программирования… Никлаусом Виртом. (Здесь должен был быть звук неожиданного сюжетного поворота.)

Слово «расширенная» в названии недвусмысленно намекает, что должна существовать и обычная, никакая, просто «форма Бэкуса-Наура» (BNF). Она действительно существует и именно она была создана американским учёным в области информатики Джоном Бэкусом и датским учёным в области информатики Петером Науром. Впоследствии Никлаус Вирт дополнил этот метаязык несколькими конструкциями, позволяющими описывать синтаксис языков более лаконично — и вот мы здесь.

Кстати, все трое в разные годы удостоены премии Тьюринга — самой престижной премии в информатике. Здесь Вам, как говорится, не тут. Пессимистично настроенный читатель скажет, что ему такого в жизни не добиться, оптимистично настроенный — что у него всё ещё впереди, но мы прислушаемся к реалистично настроенному читателю, который говорит: «Эй, ну давайте уже по существу вопроса!»

В расширенной форме Бэкуса-Наура используются такие метасимволы:

Метасимвол	Значение метасимвола
= или ::=	«Определяется как», «по определению есть».
.	Обозначает конец правила.
|	Выбор, альтернатива («либо», «или»).
{ }	Повторение конструкции, заключённой в фигурные скобки, 0 или более раз.
[ ]	Необязательная часть, т.е. конструкция в квадратных скобках используется либо 0, либо 1 раз.
( )	Используются для группировки частей определений.

Метаконстанты в РБНФ записываются в двойных кавычках " ", метапеременные — в угловых скобках < >. Если в записи метапеременной есть пробелы, вместо них записывают символы нижнего подчёркивания.

Попробуем рассмотреть применение РБНФ на каком-нибудь несложном примере. Возьмём, скажем, десятичные числа. В их записи есть целая и необязательная дробная часть, причём если дробная часть всё же есть, то от целой она отделяется точкой или запятой (в разных странах по-разному). А вот так можно выразить правила записи десятичных чисел в РБНФ:

```
<Десятичное_число> ::= <Целая_часть> [<Дробная_часть>].
<Целая_часть> ::= <Десятичная_цифра_кроме_0> {<Десятичная_цифра>}.
<Дробная_часть> ::= ("." | ",") <Десятичная_цифра> {<Десятичная_цифра>}.
<Десятичная_цифра_кроме_0> ::= "1" | "2" | "3" | "4" | "5"
                             | "6" | "7" | "8" | "9"
<Десятичная_цифра> ::= "0" | <Десятичная_цифра_кроме_0>
```

В этом описании, включающем в себя 5 синтаксических единиц (т.е. 5 правил), использованы метапеременные <Десятичное_число>, <Целая_часть>, <Дробная_часть>, <Десятичная_цифра_кроме_0> и <Десятичная_цифра>. Обратите внимание, что в каждой синтаксической единице можно выделить левую и правую часть, разделённые метасимволом ::=. Каждая из метапеременных встретилась в левой части какой-либо синтаксической единицы ровно по одному разу — это обязательное условие для того, чтобы описание было полным.

Попробуем прочитать правила приведённого описания.

Правило первое: «Десятичное число по определению есть целая часть, за которой следует необязательная дробная часть».

Правило второе: «Целая часть по определению есть десятичная цифра, за которой следует произвольное количество (от 0 до бесконечности) десятичных цифр».

Правило третье: «Дробная часть по определению есть точка или запятая, за которыми следует десятичная цифра, а затем ещё произвольное количество (от 0 до бесконечности) десятичных цифр».

Правило четвёртое: «Десятичная цифра (кроме 0), по определению есть символ 1, или символ 2, или символ 3… и так далее… или символ 8, или символ 9».

Правило пятое: «Десятичная цифра по определению есть символ 0 или десятичная цифра (кроме 0)».

Начнём с одной-единственной метапеременной <Десятичное_число> и начнём шаг за шагом заменять самую первую метапеременную в строке правой частью соответствующей синтаксической единицы, учитывая встречающиеся метасимволы.

```
<Десятичное_число>
<Целая_часть> [<Дробная_часть>]
<Десятичная_цифра_кроме_0> {<Десятичная_цифра>} [<Дробная_часть>]
```

При замене метапеременной <Десятичная_цифра_кроме_0> нам придётся выбрать одну из метаконстант-цифр. Пусть это будет, например, цифра 1.

```
"1" {<Десятичная_цифра>} [<Дробная_часть>]
```

На этом шаге мы видим метасимвол { }, которые означает, что записанное внутри фигурных скобок может повториться 0 или более раз. Пусть в нашем случае это будет, например, два повторения.

```
"1" <Десятичная_цифра> <Десятичная_цифра> [<Дробная_часть>]
```

Выполним подстановку дальше (записана в круглых скобках):

```
"1" ("0" | <Десятичная_цифра_кроме_0>) <Десятичная_цифра> [<Дробная_часть>]
```

Выберем цифру 0.

```
"1" "0" <Десятичная_цифра> [<Дробная_часть>]
```

Заменим ещё одну метапеременную и выберем цифру 8.

```
"1" "0" ("0" | <Десятичная_цифра_кроме_0>) [<Дробная_часть>]
"1" "0" <Десятичная_цифра_кроме_0> [<Дробная_часть>]
"1" "0" "8" [<Дробная_часть>]
```

Метасимвол [ ] означает, что записанное в квадратных скобках можно либо использовать, либо отбросить. Решаем использовать.

```
"1" "0" "8" <Дробная_часть>
"1" "0" "8" ("." | ",") <Десятичная_цифра> {<Десятичная_цифра>}
```

Выберем в качестве разделителя целой и дробной части запятую.

```
"1" "0" "8" "," <Десятичная_цифра> {<Десятичная_цифра>}
```

Продолжим заменять. Пусть у нас первая цифра после запятой будет цифрой 7.

```
"1" "0" "8" "," "7" {<Десятичная_цифра>}
```

Снова видим метасимвол { }, означающий возможность повторения 0 или более раз. На этот раз решим повторить заключённое в фигурные скобки 0 раз, т.е. просто пропустить.

```
"1" "0" "8" "," "7"
```

Мы получили строку, состоящую только из метаконстант. По определению метаконстанта задаёт точную запись какой-то части текста. В нашем случае полученная последовательность метаконстант задаёт число «108,7».

Обратите внимание, что по такому же принципу, но принимая другие решения в отношении метасимволов, можно получать записи произвольных целых и дробных десятичных чисел. В этом и заключается преимущество метаязыков по сравнению со словесным описанием синтаксиса и описанием синтаксиса на примерах: метаязыки исключают двусмысленность и, при правильном составлении описания, позволяют задать абсолютно все возможные способы записи тех или иных конструкций языка.

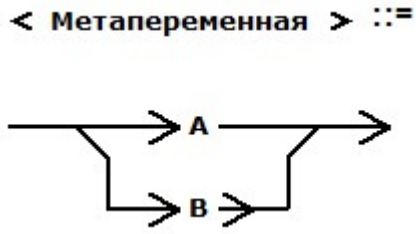
РБНФ (иногда с небольшими изменениями) применяется во многих инструментах, позволяющих автоматизировать разработку отдельных частей компиляторов.

Синтаксические диаграммы

Основным преимуществом РБНФ является то, что получающиеся описания представляют собой обычный текст, который легко может быть введён с клавиатуры и обработан программно. Это же является и недостатком: человеку намного проще воспринимать графическую информацию, чем текст, поэтому РБНФ-описания не могут похвастаться наглядностью, а пропущенные или избыточные метасимволы — это ошибки, которые легко допустить и не заметить при составлении описания.

Решает эту проблему ещё один популярный метаязык — синтаксические диаграммы. Как следует из названия, описания на этом языке будут представлять собой не текст, и это основное отличие от РБНФ. Кроме того, в синтаксических диаграммах метаконстанты не заключаются в двойные кавычки (в отличие от РБНФ здесь не возникает проблемы определения того, где начинается и заканчивается их запись), а метасимволы не используются совсем. Роль метасимволов выполняют направления линий.

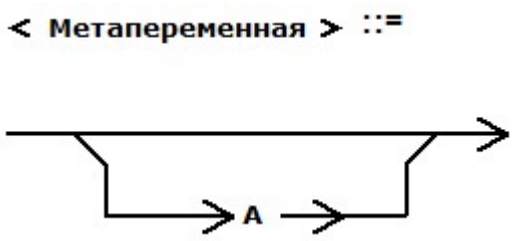
Для начала рассмотрим базовые конструкции. Так обозначается альтернатива, выбор из двух вариантов:



То же самое в РБНФ выглядело бы так:

```
<Метапеременная> ::= "A" | "B".
```

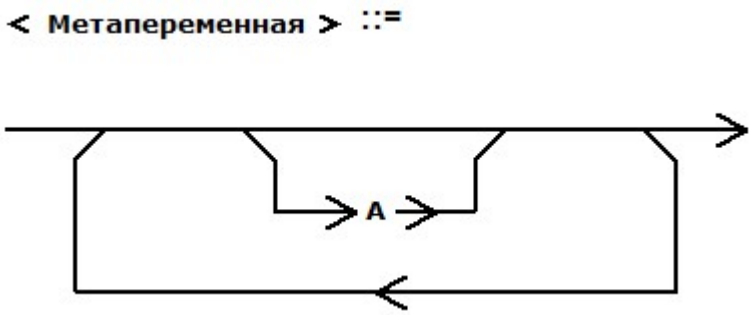
Необязательные части обозначаются ещё проще:



То же самое в РБНФ выглядело бы так:

```
<Метапеременная> ::= ["A"].
```

Для организации повторений используется более сложная конструкция:

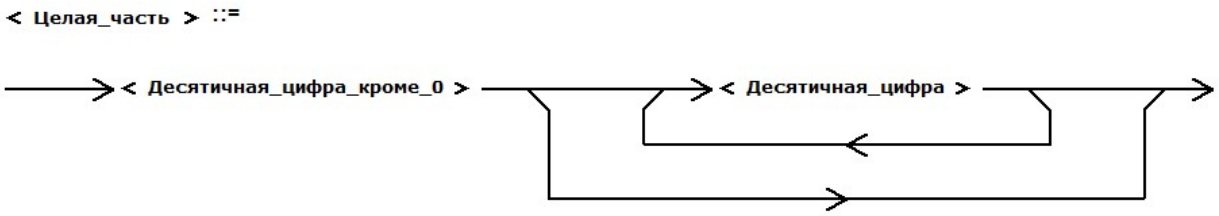


Приведённая синтаксическая диаграмма в точности эквивалентна следующей записи в РБНФ:

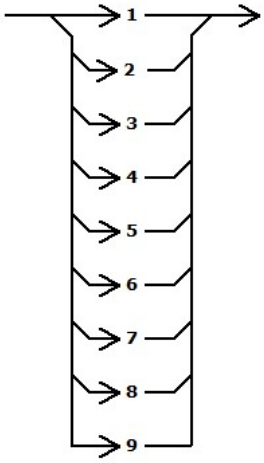
```
<Метапеременная> ::= {"A"}.
```

Следует обратить внимание на то, что в местах соединения линии входят и выходят под углом в 45 градусов, с учётом предполагаемого направления дальнейшего движения. Это необходимо, чтобы избежать неоднозначностей при прочтении диаграмм.

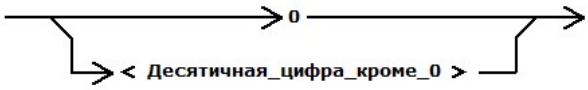
Рассмотрим применение синтаксических диаграмм на том же примере, что и РБНФ, — записи целых и дробных десятичных чисел.



< Десятичная_цифра_кроме_0 > ::=



< Десятичная_цифра > ::=



Обратите внимание, что в синтаксической диаграмме для метапеременной <Дробная_часть> удалось упростить синтаксическую единицу, оставив только обратную линию при метапеременной <Десятичная_цифра>. В остальном синтаксические диаграммы соответствуют ранее приведённому РБНФ-описанию.

Для чтения синтаксических диаграмм достаточно следовать направлениям линий, выполняя подстановки и принимая решения на разветвлениях.

Дополнительные вопросы

1. Приведённые примеры описаний для десятичных чисел с необязательной дробной частью не задают запись некоторых чисел с дробной частью. Каких?

2. Как необходимо доработать описания, чтобы учесть этот случай?

3. Предложите описание синтаксиса для математических выражений, состоящих из чисел и знаков 4 базовых операций — сложения, вычитания, умножения и деления.

4. Доработайте описание из п. 3 для поддержки круглых скобок.

5. Доработайте описание из п. 4 для поддержки тригонометрических функций, а также натурального и десятичного логарифмов: sin x, cos x, tg x, ctg x, ln x, lg x.