# Лицензия

ЧАСТЬ 1

# 1 Арифметика целых чисел

Когда у человека возникла потребность в счете, появились первые числа. Такие числа в математике называют натуральными. Натуральный ряд чисел – это числа 1, 2, 3, … Он образует бесконечное множество натуральных чисел $\mathbb{N}$, поэтому наибольшего натурального числа не существует. А наименьшее натуральное число равно единице.

Если к множеству $\mathbb{N}$ добавить ноль, то получится множество неотрицательных чисел. Отрицательные числа расширяют $\mathbb{N}$ до множества целых чисел $\mathbb{Z}$.

Над множеством $\mathbb{Z}$ определен ряд арифметических операций: сложение, вычитание, умножение, целочисленное деление с остатком и изменение знака.

В математике множества $\mathbb{N}$ и $\mathbb{Z}$ бесконечны. Но в компьютерах (и в языках программирования) они ограничены размерами памяти, отводимыми на представление целых чисел. Натуральные числа и ноль представляются в прямом двоичном коде, поэтому диапазон их представления составляет $0 \le n \le 2^m-1$, где m – количество бит, отводимых под представление числа n. Отрицательные числа представляются в дополнительном коде, и диапазон их представления составляет $-2^m \le n < 0$.

В компьютерной арифметике преимущества работы с целыми числами заключаются в том, что результаты арифметических операций всегда точны, а время выполнения обычно меньше, чем при работе с числами других разновидностей. Главный недостаток - необходимость следить за тем, чтобы значения не выходили за пределы отведенного им диапазона.

## 1.1 Целые типы в PascalABC.NET

Перед тем, как начать работу, мы оцениваем отведенное нам рабочее пространство с тем, чтобы определить, разместится ли в нем все необходимое. Такую же оценку должен сделать компилятор перед созданием машинного кода. С этой целью программист указывает в своей программе тип каждого элемента данных. Тип определяет множество значений, которые может принимать элемент, а также множество производимых с ним операций. Зная тип, компилятор может определить размер памяти, который следует отвести под данные и особенности операций, которые с этими данными надо произвести. Типы целых чисел, приведены в таблице 1.1.

Тестирование автором в среде Windows показало, что процессор Intel Core 2 Duo E8400 обеспечивает наибольшую скорость выполнения арифметических операций при работе с данными типа **integer** и **cardinal**. Использование типов данных **byte** (**shortint**) ухудшает производительность на 15-20%, тип данных **int64** (**uint64**) ухудшает производительность на 10-20%, а тип **word** (**smallint**) показал двукрат-

ное драматическое снижение производительности. Возможно, в других условиях будут получены другие данные, но это лишь доказывает, что всегда полезно знать особенности системы, в которой исполняется программа. Особую актуальность такое знание приобретает при выполнении встречающихся в различных конкурсных заданиях условий написать программу, оптимальную по времени исполнения и/или памяти.

| Длина, байт | Без знака | Со знаком |
|---|---|---|
| 1 | **byte**<br>0 .. 255 | **shortint**<br>-128 .. 127 |
| 2 | **word**<br>0 .. 65 535 | **smallint**<br>-32 768 .. 32 767 |
| 4 | **longword**, **cardinal**<br>0 .. 4 294 967 295 | **integer**, **longint**<br>-2 147 483 648 .. 2 147 483 647 |
| 8 | **uint64**<br>0 .. 18 446 744 073 709 551 615 | **int64**<br>-9 223 372 036 854 775 808 .. 9 223 372 036 854 775 807 |
| переменная | | **BigInteger**<br>Ограничено памятью компьютера |

Таблица 1.1. Целые типы PascalABC.NET

### 1.1.1 Тип BigInteger

Позволяет работать с данными, имеющими в записи практически неограниченное количество цифр. Для работы с числами в таком формате используется поразрядная арифметика (примерно, как мы считаем «в столбик»), поэтому скорость работы с данными типа **BigInteger** во много раз ниже, чем с прочими. Любые целочисленные данные могут быть без потерь преобразованы к типу **BigInteger**. Обратное преобразование к другим целым типам запрещено, поскольку оно может сопровождаться потерей данных. Подробнее о работе с этим типом данных будет рассказано в части 7. Пока отметим, что данные типа **BigInteger** могут включаться в выражения наряду с прочими типами.

## 1.2 Константы

В программе могут присутствовать величины, значения которых неизменяемы. Они могут быть никогда не изменяемыми (например, в 1 метре 100 см и поэтому коэффициент для перевода метров в сантиметры всегда равен 100), или неизменяемыми в пределах программы (требуется найти сумму 10 случайных чисел из диапазона от -5 до +20). Такие неизменяемые величины называют константами.

Константы описываются перед программным блоком **begin** … **end** в так называемом ***разделе описания констант***, начинающемся служебным словом **const**.

Каждая константа описывается в виде

```
имя константы = значение; // именованая константа с неявным типом
```

или

```
имя константы: тип = значение; // типизированная константа
```

В диалекте языка Turbo Pascal (Borland Pascal) значение типизированной константы разрешено изменять присваиванием. В Borland Delphi такое изменение по умолчанию запрещено, но его можно разрешить директивой компилятора {$J+}. Free Pascal, наоборот, по умолчанию разрешает изменять значения типизированных констант, но это изменение можно запретить директивой компилятора {$J-}. PascalABC.NET **категорически запрещает** менять значение констант, в том числе типизированных.

Основное назначение констант – задавать их значения в одном, общем для всей программы месте и пресекать любые попытки программиста изменить эти значения. Подумайте, хорошо ли будет, если где-то в программе случайно изменить ранее объявленное значение числа π?

В языке PascalABC.NET константы несколько теряют свою значимость по сравнению с базовым Паскалем (это будет понятно при знакомстве с динамическими массивами в части 5), но их польза несомненна.

Кроме констант, значение которых определяет программист, имеется некоторое количество предопределенных констант, т.е. констант, значения которых компилятору уже знакомо и описывать их не нужно. Для целых типов предопределены следующие константы: MaxShortInt, MaxByte, MaxSmallInt, MaxWord, MaxInt, MaxLongWord, MaxInt64 и MaxUInt64. Их имена получаются путем приписывания Max к имени соответствующего типа, а значения равны максимально допустимым для этого типа значениям в соответствии с таблицей 1.1. Все эти имена помнить совсем необязательно, они есть в Справке, а кроме того, для каждого целочисленного типа определен набор констант вида T.V, где T – тип, а V описывает, что представляет константа:
`T.MaxValue` – максимальное значение типа T;
`T.MinValue` – минимальное значение типа T.

Например, int64.MaxValue, как и MaxInt64, определяет максимальное значение для типа int64, равное 9 223 372 036 854 775 807.

Все константы, о которых говорилось выше, имеют свои имена. Именованные константы удобны тем, что имя запомнить обычно намного проще, чем числовое значение. Например, значительно удобнее использовать константу РадиусЗемли, чем величину 6356863, увидев которую, не сразу догадаешься, что это такое.

```
// p0102
const
  Длина = 400;
  Ширина = 730;
  Высота = 142;

begin
  Println('Объем параллелепипеда равен', Длина * Ширина * Высота)
end.
```

Наряду с именованными константами существуют константы неименованные – ***литералы***. Целочисленный литерал (в переводе с латыни – буквальный) изображает значение числовой величины. В приведенном выше примере 400, 730 и 142 – литералы. В программах литералы используют повсеместно.

Числовое значение представляется литералом в привычном виде: последовательностью цифр, которая может впереди иметь знак плюс или минус. Считается, что число записано в десятичной системе счисления. Можно также записать шестнадцатеричное число, для чего первым символом указывают знак денежной единицы $. В этом случае знак числа писать нельзя, поскольку шестнадцатеричная запись отражает внутреннее представление числа.

Вы ошибетесь, если решите, что целочисленная константа, тип которой не указан явно, получит тип, который достаточен, чтобы вместить значение, указанное литералом. На самом деле, тип будет совпадать с типом литерала, использованным в качестве значения константы, а этот тип определяется в соответствии с таблицей 1.2.

| Минимальное значение | Максимальное значение | Тип |
|---|---|---|
| -2 147 483 648 | 2 147 483 647 | **integer** |
| -9 223 372 036 854 775 808 | 9 223 372 036 854 775 807 | **int64** |
| 9 223 372 036 854 775 808 | 18 446 744 073 709 551 615 | **uint64** |
| < -9 223 372 036 854 775 808 | > 18 446 744 073 709 551 615 | сообщение об ошибке |

Таблица 1.2. Типы данных, назначаемые целочисленным литералам.

Кроме литерала, значение константы может определяться ***арифметическим выражением*** (глава 1.4), результатом вычисления которого должна быть целочисленная величина. В PascalABC.NET при определении значения константы посредством арифметического выражения запрещается использовать пользовательские функции, а значения переменных в выражении должны быть известны компилятору.

Встретив в программе описание константы, компилятор устанавливает ее тип, отводит в памяти место, достаточное для размещения значения константы, вычисляет значение константы и помещает его отведенную память. Вся работа совершается на стадии компиляции, поэтому к моменту начала выполнения программы значения констант уже находятся на отведенных им местах. Этим

объясняется причина запрета попыток изменить значение константы при выполнении программы.

## 1.3 Переменные

Кроме констант, в программе могут присутствовать величины другого рода, значение которых может изменяться в процессе выполнения программы. Такие величины называются ***переменными***. Каждой переменной программист назначает имя (***идентификатор***), с помощью которого затем оперирует с ней. Как и константа, переменная имеет тип. Типы у целочисленных переменных точно такие же, как у целочисленных констант, поэтому информация из таблицы 1.1. верна и для переменных. Как и константе, компилятор отводит переменной место в памяти компьютера, но содержимое этой области памяти разрешено менять.

Перед тем, как первый раз использовать переменную, ее необходимо описать, дав возможность компилятору установить тип этой переменной.

> PascalABC.NET рекомендует описывать переменные непосредственно перед их использованием, а не в отдельном разделе описания переменных, как это требовалось в базовом Паскале. Переменные, описанные в некотором блоке, за его пределами не существуют.

Описание переменной имеет вид

```
var ИмяПеременной: тип;
```

где тип мы пока будем выбирать из таблицы 1.1.

Если нужно описать несколько переменных одного типа, их имена перечисляются списком через запятую:

```
var ИмяПеременной1, ИмяПеременной2, … ИмяПеременнойN: тип;
```

Недопустимо смешивать в одном списке var описания переменных различных типов.

При описании переменной можно присвоить ей начальное значение (это называется ***инициализацией***), но лишь одной для каждого var:

```
var Имя переменной: тип := значение;
```

Конструкция := в языке Паскаль носит название знака ***операции присваивания***. Она понимается следующим образом: следует вычислить значение выражения, помещенного справа от знака операции присваивания и поместить его в переменную, имя которой указано слева от этого знака.

В случаях, когда тип переменной можно установить из указанного или вычисленного значения, PascalABC.NET позволяет делать автовыведение типа. Чтобы использовать автовыведение типа, в описании переменной тип не указывается. Как и

в случае с константами, тип которых явно не указан, он устанавливается компилятором в соответствии с таблицей 1.2.

```
var Имя переменной := значение;
```

Примеры описаний переменных.

```
var a, b, gamma, w15: integer; // описание переменных списком
var bt: byte; // описание одной переменной
var n:word := 18; // описание типа, совмещенное с инициализацией
var s := 0; // описание с автовыведением типа (integer)
var MyBytes := $C7; // шестнадцатиричное значение с типом integer
```

Имя переменной в PascalABC.NET может иметь практически любую длину и образуется из букв, цифр и знака подчеркивания, но с цифры оно начинаться не может. Прописные и строчные буквы не различаются, поэтому имена proba, PrObA и PROBA эквивалентны. Буквы не обязаны быть только латинскими – допускаются любые буквы Unicode – русские (кириллица), западноевропейская латиница, греческие и т.д. Поэтому появление переменной с именем Уголβ вопросов у компилятора не вызывает. Недопустимо использовать в качестве имен ключевые слова языка, но и тут есть выход: в таких случаях перед именем ставится ***экранирующий символ*** &, например &begin.

При описании переменной целого типа, объединенном с инициализацией, в качестве значения может указываться произвольное арифметическое выражение, результатом вычисления которого должна быть целочисленная величина. Если это условие нарушить, то при наличии описания типа компилятор зафиксирует ошибку, а при отсутствии описания переменная получит не тот тип, который программист ожидал.

Имеется еще одна, мощная разновидность описания переменных, объединенного с присваиванием начального значения и автовыведением типов. Она основана на так называемом ***кортежном присваивании***.

```
var (Имя1, Имя2, … ИмяN) := (Значение1, Значение2, … ЗначениеN);
```

Здесь переменная Имя1 получает Значение1 (и соответствующий тип), Имя2 получает Значение2 и т.д.

Конструкция, записанная в круглых скобках, называется ***кортежем*** (раздел 5.1.4), отсюда и происходит название этого вида множественного присваивания.

```
var (a, b, c, i) := (132, -58, 0, 1);
```

Обратите внимание, что указывать тип переменных в кортежном списке нельзя. Конструкция вида

```
var (a, b, c:int64, i:byte) := (132, -58, 0, 1);
```

будет забракована компилятором. При необходимости явно указать типы можно либо отказаться от кортежного присваивания, либо воспользоваться ***явным приведением типа*** (глава 1.7) в правой части и сделать описание примерно таким:

```
var (a, b, c, i) := (132, -58, int64(0), byte(1));
```

Использование неинициализированных переменных алгоритмически неверно и часто приводит к плохо обнаруживаемым ошибкам. Чтобы помочь начинающим пользователям, в PascalABC.NET любая числовая переменная, которая при описании не получила начального значения, инициализируется нулем. Тем не менее, не нужно на это полагаться Хороший стиль программирования предполагает, что программист сам проводит необходимую инициализацию переменных.

# 1.4 Арифметические выражения

В языках программирования под термином «выражение» понимают набор из имен переменных, констант, знаков операций, скобок и имен функций. Арифметическое выражение является аналогом математической формулы и результатом его вычисления будет число. Если это число будет целым, то говорят о целочисленном арифметическом выражении. Частными случаями выражения являются уже рассмотренные константы и переменные.

## 1.4.1 Арифметические операции

Константы и переменные в арифметическом выражении могут связываться между собой при помощи знаков арифметических операций. При этом образуются конструкции вида

```
A ЗнакОперации B
ЗнакОперации B
```

Здесь A и B называются ***операндами***, а знак операции принято называть ***операцией***. Если операция используется с двумя операндами, она называется ***бинарной***. Существуют также операции с одним операндом, называемые ***унарными***. Есть также операция с тремя операндами, она называется **тернарной**, но в PascalABC.NET используется термин ***условная операция*** (глава 3.3).

К арифметическим операциям относятся сложение (знак операции + ), вычитание ( - ) и умножение ( * ). Привычный знак деления ( / ) мы пока использовать не будем, потому что результат этой операции в Паскале не является целочисленным. Деление нацело выполняет операция **div**. Еще одна операция, **mod**, дает (в программировании принято говорить «***возвращает***») остаток от целочисленного деления. Арифметическая унарная операция фактически одна – изменение знака числа, для чего перед операндом указывается знак минус. Может быть также указан и плюс, но он не выполняет никаких действий.

Операции возведения в степень в Паскале нет! Для целых чисел есть только ***функция*** возведения в квадрат Sqr(n). Отметим, что часто бывает удобнее и короче написать n*n, чем Sqr(n). В куб возводим, записывая n*n*n, в четвертую степень можно возвести, записав Sqr(Sqr(n)) и т.д.

Рассмотрим программу.

```
begin
   var (a, b) := (30, 8);
   var c := a + b;
   var d := c * a + 2 * b;
   var (e, f) := (a div b, a mod b);
   var g := - e + f;
   Println(a, b, c, d, e, f, g)
end.
```

Вначале a получает значение 30, b получает значение 8 (тип обоих **integer**). Затем вычисляется значение a + b. 30 + 8 = 38. Переменная c получает значение 38. А что даст автовыведение типа? Складываются значения типов **integer**. Ожидаемо, что тип результата тоже будет **integer**. Далее, 38 * 30 + 2 * 8 = 1156 и это значение запоминается в d. Здесь тоже понятно, какой будет тип – **integer**. Вычисляется выражение a **div** b. 30 делим нацело на 8. Фактически, выделяем целую часть простой дроби 30 / 8, получая число 3, которое помещается в переменную e. Тип будет снова **integer**. Вычисляем a **mod** b. Это остаток от деления 30 / 8. Частное мы уже нашли, оно равно 3, поэтому 30 - 3 * 8 = 6. Значение 6 отправляется в f, по-прежнему неся с собой тип **integer**. Переходим к строке с описанием переменной g. Вот он – унарный минус – стоит перед a. Меняем знак e, получая -3 и это значение складываем со значением f, равным 6, получая в результате 3. Оно отправляется в g, а тип его, конечно же, все тот же: **integer**.

Если написать кортежное присваивание

```
var (c, d) := (a + b, c * a + 2 * b);
```

результат окажется не тем, который ожидается. Здесь d зависит от c, но невозможно угадать, что будет вычисляться раньше, c или d. Поэтому компилятор в данном случае зафиксирует ошибку и выдаст сообщение «Неизвестное имя 'c'».

## 1.4.2 Приоритет арифметических операций

Когда мы вычисляли значение выражения c*a+2*b, то не задумываясь нашли значений произведений c*a и 2*b, а только затем их сложили между собой. Почему так? Разве нельзя сначала найти значение c*a, прибавить к нему 2, и лишь потом умножить результат на b? Дело в том, что мы еще в начальной школе усвоили правило арифметики: умножение делается раньше сложения. Чтобы не путаться во множестве правил, в программировании введено понятие ***приоритета операций***.

Приоритет операции – это некоторое целое число. Чем оно меньше, тем приоритет выше, тем раньше выполняется операция. Операции, имеющие одинаковый прио-

ритет, выполняются в естественном порядке следования, слева направо. Для изменения порядка выполнения операций служат круглые скобки, имеющие наивысший приоритет. Знание приоритета операций позволяет избежать нагромождений из скобок и обеспечить правильный порядок выполнения операций.

В описании каждого языка программирования имеется таблица приоритетов всех имеющихся в языке операций. Нам пока достаточно следующих знаний:

- унарные операции + и – , а также функция Sqr() имеют приоритет 1;
- операции *, **div**, **mod** имеют приоритет 2;
- бинарные операции + и – имеют приоритет 3.

Если приоритет нескольких подряд идущих операций одинаков, они выполняются в порядке слева направо. Об этом часто забывают.

## 1.4.3 Игры с унарным плюсом и минусом

Пусть дана следующая программа

```
begin
   var (a, b, c) := (3, -4, 8);
   var y := -2 * a + 5 * b + - - - + - - + + + - 2 * c;
   Println(y)
end.
```

Является ли такое «дикое» выражение, инициализирующее y, допустимым?

Для ответа на этот вопрос попытаемся разбить это выражение на отдельные операции при помощи расстановки скобок. Также подставим вместо переменных их значения. Не забываем, конечно, о приоритете операций.

```
-2*a+5*b+---+--+++-2*c
((-2)*3) + (5*(-4)) +---+--+++-2*8
(-6)+(-20) + (---+--+++-)2*8 выделяем унарные операции
-26 + (---+--+++-)2*8
Унарные плюсы опускаем – они ничего не меняют
-26 + (------)2*8
Осталось 6 унарных минусов; это три пары. Каждая пара унарных минусов дает
унарный плюс.
(-26) + (+++)8*2
Снова опускаем унарные плюсы
(-26) + 8*2
(-26) + 16
-26+16
-10
```

Можно утверждать, что переменная y будет инициализирована значением -10 с автовыведением типа **integer**. Кажется сложным? Может быть. Но если понять, как

это делается, в дальнейшем не будет проблем, связанных с унарными операциями любого типа и над любыми данными.

## 1.4.4 Стандартные целочисленные функции

Арифметическое выражение может содержать вызовы ***функций*** (глава 4.1).

Под функцией в программировании понимают некоторый самостоятельный фрагмент программы, имеющий имя, к которому можно обращаться из других мест программы (***вызывать*** функцию) путем упоминания этого имени. Часть функций компилятор «знает» и их называют стандартными или встроенными. Другую часть пользователь при необходимости может ***подключить*** из имеющихся внешних файлов-библиотек («модулей»), либо запрограммировать самостоятельно (***пользовательские функции***).

Обращение к функции (ее вызов) состоит в записи имени функции, за которым в круглых скобках следует список передаваемых ей параметров (***аргументов функции***), на основе которых будет вычисляться значение. Найденное значение подставляется на место вызова функции. Параметры отделяются друг от друга запятыми. Если параметров нет, круглые скобки после имени функции тоже можно не указывать.

Рассмотрим некоторые из стандартных функций PascalABC.NET

`Abs(n)` – возвращает абсолютное значение аргумента n
`Max(m, n)` – возвращает максимальное из значений m, n
`Min(m, n)` – возвращает минимальное из значений m, n
`Random(n)` – возвращает целое случайное число из диапазона 0 .. n-1
`Random(m, n)` – возвращает целое случайное число из диапазона m .. n
`Random2(n)` – возвращает кортеж из двух целых случайных чисел в диапазоне 0 .. n;
`Random2(m, n)` – возвращает кортеж из двух целых случайных чисел в диапазоне m .. n;
`Random3(n)` – возвращает кортеж из трех целых случайных чисел в диапазоне 0 .. n;
`Random3(m, n)` – возвращает кортеж из трех целых случайных чисел в диапазоне m .. n;
`Sign(a)` – возвращает -1 при a<0, 0 при a=0 и 1 при a>0;
`Sqr(n)` – возвращает квадрат n (следите за типом, может вернуть **int64** !!!).

Теперь мы можем составлять достаточно сложные целочисленные выражения и проводить с их помощью инициализацию описываемых переменных . Этого вполне достаточно, чтобы начать писать несложные программы, производящие вычисления и выводящие полученные результаты.

```
begin // p010404
  var (Длина, Ширина) := Random2(100, 400);
  var (ПериметрПрямоугольника, ПлощадьПрямоугольника) :=
      (2 * (Длина + Ширина), Длина * Ширина);
  var (Катет1, Катет2) := Random2(10, 90);
  var КвадратГипотенузы := Sqr(Катет1) + Sqr(Катет2);
  var МаксимумТрех := Max(Random(30), Max(Random(25),
      Random(21)));
  Println(Длина, Ширина, ПериметрПрямоугольника,
      ПлощадьПрямоугольника);
  Println(Катет1, Катет2, КвадратГипотенузы, NewLine,
      МаксимумТрех)
end.
```

Вывод будет выглядеть так:

```
373 321 239466 119733
81 88 14305
 29
```

Представим себя на месте пользователя, для которого мы написали такую программу. Как ему понять, где что выведено? Текста программы он не видит, а даже если бы и увидел, текст ему мало что даст. Это приводит нас к пониманию следующего: мало составить алгоритм и написать программу, - надо еще позаботиться о том, чтобы эта программа обеспечивала удобный для пользователя ввод и вывод информации.

Набор программных средств и приемов программирования, формирующих взаимодействие пользователя с программой, образует так называемый ***пользовательский интерфейс***. Создание эффективных пользовательских интерфейсов – отдельная область конструирования программ, которая выходит за рамки данной книги. Но какие-то, пусть минимальные, удобства для пользователя программа обеспечить обязана. Например, вывести кроме набора чисел необходимые пояснения. Сделать это совсем несложно, достаточно вставить в нужных местах списка выводимых значений поясняющий текст в одинарных кавычках.

```
Println('Длина=', Длина, 'ширина=', Ширина, 'периметр=',
        ПериметрПрямоугольника, 'площадь=', ПлощадьПрямоугольника);
```

Соответствующая строка примет следующий вид

```
Длина= 276 ширина= 313 периметр= 172776 площадь= 86388
```

Оператор Println чаще используют при выводе значений переменных без поясняющего текста, поскольку он разделяет выводимые элементы пробелом. Если разделители мы хотим указать сами, удобнее использовать оператор Writeln (или Write, если после вывода строку переводить не надо). Этот оператор не выводит ничего помимо указанного в списке.

```
Writeln('Длина=', Длина, ', ширина=', Ширина, ', периметр=',
        ПериметрПрямоугольника, ', площадь=', ПлощадьПрямоугольника);
```

Вывод станет еще нагляднее

```
Длина=352, ширина=299, периметр=210496, площадь=105248
```

Остановимся на том, как работает эта программа, опуская некоторые подробности ввиду достаточно информативных имен переменных.

Функция Random2(100, 400) возвращает кортеж из двух случайно выбранных из диапазона [100;400] целых чисел. Кортежное присваивание инициализирует переменную Длина первым из элементов кортежа, а переменную Ширина – вторым. В следующей строке описаны две переменные, которые также инициализируются кортежным присваиванием. Переменные Катет1 и Катет2 инициализируются кортежным присваиванием на основе функции Random2, а переменная МаксимумТрех при инициализации дважды обращается к стандартной функции Max.

Ниже приведены примеры записи целочисленных арифметических выражений.

| Математическая запись | Запись на PascalABC.NET | Ошибочная запись на PascalABC.NET |
|---|---|---|
| $-\frac{a \cdot b}{c \cdot d}$ | -(a*b **div** (c*d)) | -a*b **div** c*d |
| $\frac{a+b}{c+d} \times \frac{1}{e}$ | (a+b) **div** (c+d) **div** e | (a+b)/(c+d)*1/e |
| $a^5 + b^8$ | a*Sqr(a*a)+Sqr(Sqr(b*b)) | |

Рассмотрим текст еще одной программы, а потом сравним его с программой на базовом Паскале. Чтобы не было «мучительно больно» за потраченное на изучение PascalABC.NET время.

```
begin
   var (a, b) := Random2(-5, 20);
   Println(a, b);
   Writeln('Минимум=', Min(a, b))
end.
```

При всей своей краткости, приведенная программа делает не так уж мало. Вначале в кортежном присваивании переменные a и b инициализируются с помощью функции получения двух случайных чисел в диапазоне [-5;20]. Полученные значения выводятся на монитор. Затем вычисляется и выводится минимальное значение из a и b. Вывод выглядит следующим образом

```
11 -4
Минимум=-4
```

Теперь рассмотрим аналогичную программу на базовом Паскале. Конечно, ее тоже можно набрать и выполнить в PascalABC.NET, поскольку его язык обеспечивает высокую совместимость с базовым Паскалем.

```
var
  a, b, min: integer;

begin
  Randomize;
  a := Random(26) - 5;
  b := Random(26) - 5;
  Writeln(a, ' ', b);
  if a < b then min := a
  else min := b;
  Writeln('Минимум=', min)
end.
```

# 1.5 Оператор присваивания

Говоря об описании переменных, объединенном с их инициализацией, удобно попутно познакомиться с оператором присваивания. Он получается, если убрать из описания переменной с автовыведением типа служебное слово **var**.

Оператор присваивания используется для ранее описанных переменных и позволяет присвоить им некоторое значение. Его структура очень проста

```
Имя переменной := выражение;
```

Есть и кортежное присваивание

```
(Имя1, Имя2, … ИмяN) := (Выражение1, Выражение 2, … Выражение N);
```

Работа оператора присваивания состоит в вычислении значения выражения в правой части и присваивания этого значения имени переменной в левой части. Тип значения, полученного в выражении, должен совпадать или быть ***автоматически приводимым*** к типу переменной. Это означает, что мы не обязаны указывать в программе, как именно производить преобразование типа.

> Для целочисленных значений все типы, кроме BigInteger, автоматически приводимы путем копирования нужного количества байт без какого-либо анализа и это может оказаться неисчерпаемым источником труднонаходимых ошибок.

Например, корректно написать

```
var a := 35*40+283; // значение 1683, тип integer, 4 байта
var b : int64 := a*a; // значение 2832489, тип int64, 8 байт
```

Но вот пример, если сказать мягко, «неожиданности»:

```
begin // p0105
  var a: int64 := 546 * 546 * 546; // должно быть 162 777 336
  a := a * a; // должно быть 26 494 507 823 224 896
  var b: integer := a * a; // то же, что и выше, но 4 байта мало
  Println(a, b)
end.
```

При выводе получаем

```
26494507823224896, -1259597824
```

Первое значение совершенно правильное – это $(546^3)^2 = 546^6$, а второе – результат попытки поместить это значение в переменную, для которой отведено количество памяти, недостаточное для размещения такого числа. В погоне за эффективностью компиляторы строят программу, бесконтрольно пересылающую нужное количество байт и в данном случае из 8 байт будет переслано только 4.

Ошибки, связанные с потерей разрядов в целочисленной арифметике, к сожалению, нередки и трудно находимы. Об этом начинающему программисту нужно помнить и быть особенно внимательным, используя «короткие» типы данных длиной 1 и 2 байта. В PascalABC.NET вероятность ошибок несколько снижена вследствие использования по умолчанию типа **integer** с длиной 4 байта, а вот в базовом Паскале тип **integer** был двухбайтным (из-за 16-битной архитектуры компьютеров прошлых лет), что нередко порождало ошибки вследствие выхода значений за величину 32767.

Но нет ли тут нарушения? Нет. В стандарте языка Паскаль (есть и такой, но строго им никто не пользуется) сказано, что представление типа **integer** должно обеспечить размещение целочисленных значений в диапазоне [-32768 ; 32767]. Но это вовсе не является запретом на размещение в типе **integer** значений большей величины.

# 1.6 «Сюрпризы» целочисленной арифметики

При вычислении выражений компилятор заводит внутренние переменные в соответствии с типом операндов. Рассмотрим пример с литералами.

```
begin
  var a: int64 := 125 * 125 * 125 * 125 * 125;
  Println(a)
end.
```

Полученное значение $125^5$ = 30 517 578 125 вполне укладывается в диапазон **int64**, но при попытке компиляции этой программы мы получим сообщение «Переполнение в арифметической операции». Объяснение причины достаточно простое. Литерал 125 оценивается, как имеющий тип **integer**. Каждая операция умножения строится по схеме **integer** := **integer** * **integer**. На последнем (четвертом) умножении значение произведения выйдет за пределы MaxInt = 2 147 483 647, что обнаружится компилятором. Что же, в этот раз повезло! Но вот другой пример, с переменными.

```
begin
  var n := 125;
  var a: int64 := n * n * n * n * n;
  Println(a)
end.
```

В отличие от предыдущего примера, вместо литерала здесь используется переменная. И при компиляции переполнение не обнаруживается, поскольку компилятор «не знает» значения n – оно появится только после выполнения присваивания, что произойдет только на стадии выполнения программы. К сожалению, при выполнении программы переполнение в целочисленной арифметике не контролируется – так уж устроены процессоры персональных компьютеров. В результате получим неверный результат 452807053. Хорошо еще, если это и вся программа. А если нет и найденное значение a, не выводясь, используется в дальнейших вычислениях? Получаем труднонаходимую ошибку.

И еще один пример. Теперь объявим n не переменной, а константой.

```
const
  n = 125;

begin
  var a: int64 := n * n * n * n * n;
  Println(a)
end.
```

Компилятор подставляет вместо константы литерал 125. Результат мы уже обсудили раньше: сообщение о переполнении в арифметической операции на стадии компиляции. Снова компилятор нас уберег от ошибки. Убедились, что константы бывают полезными?

## 1.7 Явное приведение типа

Бывают случаи, когда программисту приходится принимать некоторые дополнительные меры к тому, чтобы результат вычисления выражения был верным.

Пусть требуется выполнить такой оператор присваивания

```
b := 152 * (integer.MaxValue + 2344322) * (23432 + 12312312);
// результат 4 030 998 575 382 838 272
```

Анализ выражения показывает, что будет получено значение, на порядки превышающее **integer**.MaxValue. Следовательно, для переменной b нам понадобится, как минимум, тип **int64**. Первая попытка начинающего программиста - решить задачу «в лоб».

```
begin
  var b: int64 := 152 * (integer.MaxValue + 2344322) * (23432 + 12312312);
  Println(b)
end.
```

При запуске такой программы будет получено уже известное нам сообщение компилятора «Переполнение в арифметической операции». Но непонятно, где такое переполнение возникает. К счастью, в программистском арсенале есть прием, называемый по известной с древних времен фразе «Разделяй и властвуй». Разобьем вычисление на части.

```
begin
  var p1: int64 := integer.MaxValue + 2344322;
  p1 := 152 * p1;
  var p2 := 23432 + 12312312;
  p1 := p1 * p2;
  Println(p1)
end.
```

И снова получаем то же самое сообщение компилятора, но теперь видно, что оно относится к оператору `var p1: int64 := integer.MaxValue + 2344322;`

Причина понятна. Константа **integer**.MaxValue имеет тип **integer**, литерал 2344322 также получает тип **integer** и это позволяет компилятору диагностировать переполнение в операции сложения.

Ситуацию можно попробовать спасти, объявив переменную (или константу) с типом **int64** и значением 2344322. Тогда компилятор будет иметь дело со сложением значений типов **integer** и **int64** и переполнения возникнуть не должно, поскольку в подобных операциях тип результата определяется типом операнда, занимающим большее количество памяти.

```
begin
  var c: int64 := 2344322;
  var p1 := integer.MaxValue + c;
  p1 := 152 * p1;
  var p2 := 23432 + 12312312;
  var b := p1 * p2;
  Println(b)
end.
```

Наша программа заработала. Более того, она выдала верный результат. Если цель была лишь в том, чтобы один раз произвести вычисление, можно на этом и закончить. Но мы попробуем улучшить программу, сделав ее нагляднее, т.е. постараемся максимально приблизить запись текста программы к исходной формуле. Это одно из проявлений хорошего стиля в программировании. Если даже через много лет понадобится внести изменения в программу, связанные с изменением формулы, вы сразу найдете нужное место. Будем считать, что место, вызывавшее проблему, найдено. И вернемся к первому варианту программы, лишь немного его подправив.

```
begin
  var c: int64 := 2344322;
  var b := 152 * (integer.MaxValue + c) * (23432 + 12312312);
  Println(b)
end.
```

И это тоже работает! Работает потому, что мы внедрили в арифметическое выражение одну величину типа **int64**, которая потребовала от компилятора сделать автоматическое приведение типов к **int64** во всех операциях, в которых хотя бы один операнд имеет такой тип.

Чтобы по-настоящему развязать руки программисту, в PascalABC.NET имеется возможность в нужный момент явно приводить тип к необходимому. Для этого используется запись вида **тип** (значение). Это не единственный способ, но пока будем пользоваться им.

```
begin
  var b := 152 * (integer.MaxValue + int64(2344322)) * (23432 + 12312312);
  Println(b)
end.
```

Не правда ли, эта программа похожа на ту, которую пытался написать начинающий программист? Только в отличие от той программы, она правильно работает. Также, можно было написать `int64(integer.MaxValue) + 2344322)`.

Явное приведение типа – мощный инструмент в руках опытного программиста. Вы избежите многих проблем, если будете в нужное время вспоминать об этой возможности помочь компилятору.

## 1.8 Ввод целочисленных данных

До сих пор мы писали программы, в которых все необходимые данные были известны заранее. Но так бывает очень редко. Типичная ситуация требует сделать нужные вычисления при заданных значениях некоторых данных. Например, мы хотим найти значение площади прямоугольника, длины сторон которого нужно указать в процессе работы программы.

Инициализация переменных или присваивание им некоторых значений в коде программы проблемы не решат. Нужна операция, позволяющая ввести нужные значения в процессе работы программы. Такой процесс называется ***вводом***.

PascalABC.NET предлагает совмещать описание переменной с вводом ее значения:

```
var ИмяПеременной := ReadInteger('Текст приглашения ко вводу');
```

Если приглашение не нужно, оператор выглядит еще проще

```
var ИмяПеременной := ReadInteger;
```

Имеется также вариант оператора с `ReadlnInteger`, но о нем мы вспомним в части 6, когда будем изучать ввод символьных данных. Пока лишь отметим, что поскольку символьные данные тут не вводятся, достаточно писать ReadInteger.

Если переменная была описана ранее, ключевое слово **var** указывать нельзя – повторное описание переменных язык запрещает.

Так вводится значение одной переменной. А если их больше? Вспомним кортежное присваивание: оно и тут имеется, только в ограниченном количестве – для двух и трех переменных.

```
var (имя1, имя2) := ReadInteger2('Текст приглашения ко вводу');
var (имя1, имя2, имя3) := ReadInteger3('Текст приглашения ко вводу');
```

В Паскале операция ввода с клавиатуры непременно должна завершиться нажатием клавиши Enter. Вводимые данные могут разделяться пробелами, но можно также в ходе ввода использовать Enter – это решает пользователь программы.

Универсальная программа для расчета площади прямоугольника могла бы выглядеть так

```
begin // p0108
  var (a, b) := ReadInteger2('Введите длину и ширину прямоугольника:');
  Println('Площадь прямоугольника равна', a * b)
end.
```

`ReadInteger2` осуществляет прием с клавиатуры двух значений типа **integer** (это подсказывает цифра 2 в его названии), а кортежное присваивание помещает эти значения в переменные a и b соответственно. Обратите внимание, что в операторе вывода указано не имя переменной, а выражение. Это нормальная практика. Если значение выражения требуется в программе единственный раз, совершенно излишне заводить переменную и делать ей присваивание – такие действия лишь тратят память компьютера и увеличивают время работы программы.

В базовом Паскале оператор ввода имел вид

```
Read(список переменных, разделенных запятыми);
```

Такой оператор полноценно поддерживается и в PascalABC.NET. Более того, можно привести примеры, когда он необходим. Это и ввод значений в количестве, большем трех, ввод значений переменных с типом, отличным от integer, ввод разнотипных значений и некоторые другие случаи.

```
begin
  var a: byte;
  var b, c: int64;
  var d: integer;
  Print('Введите значения a,b,c,d:');
  Read(a, b, c, d);
  …
end.
```

# 1.9 Инкремент и декремент

В ряде алгоритмов требуется изменять значение переменной на определенную величину, чаще всего, на единицу. Если эта величина положительная, говорят об ***инкременте*** (по-английски increment – увеличение), если отрицательная – о ***декременте*** (decrement – уменьшение).

Инкремент и декремент можно реализовать операцией присваивания, например

```
i := i + 1;
j := j - 2;
```

С точки зрения математики написаны абсурдные вещи. Но знак := – не знак равенства, это знак операции присваивания. Вначале вычисляется значение правой части и только потом оно запоминается в переменной, указанной в левой части. К текущему значению i прибавляется единица, и новое значение запоминается в качестве i. А новое значение j окажется уменьшенным на 2.

В языке Паскаль для таких операций есть более короткая запись:

```
Inc(i); // инкремент 1
Dec(j, 2); // декремент 2
```

Общий вид записи операций инкремента и декремента следующий

```
Inc(ИмяПеременной, ЗначениеИнкремента);
Dec(ИмяПеременной, ЗначениеДекремента);
```

Если значение инкремента или декремента равно единице, его можно не указывать.

Еще одну реализацию инкремента и декремента предоставляет разновидность оператора присваивания, позаимствованная из языков семейства C. Присваивание, соединенное с инкрементом или декрементом, можно записать в виде

```
i += 1;
j -= 2;
```

Операция += увеличивает значение переменной, указанной в левой части на величину, указанную в правой части. Операция -= уменьшает значение переменной, указанной в левой части на величину, указанную в правой части.

Поскольку инкремент и декремент реализуются посредством целочисленных операций сложения и вычитания, при их выполнении могут возникать ошибки, связанные с выходом значений за отведенные границы типа (***переполнение разрядной сетки***). Отметим, что также есть операция *=, работающая аналогично +=, но производящая не сложение, а умножение.