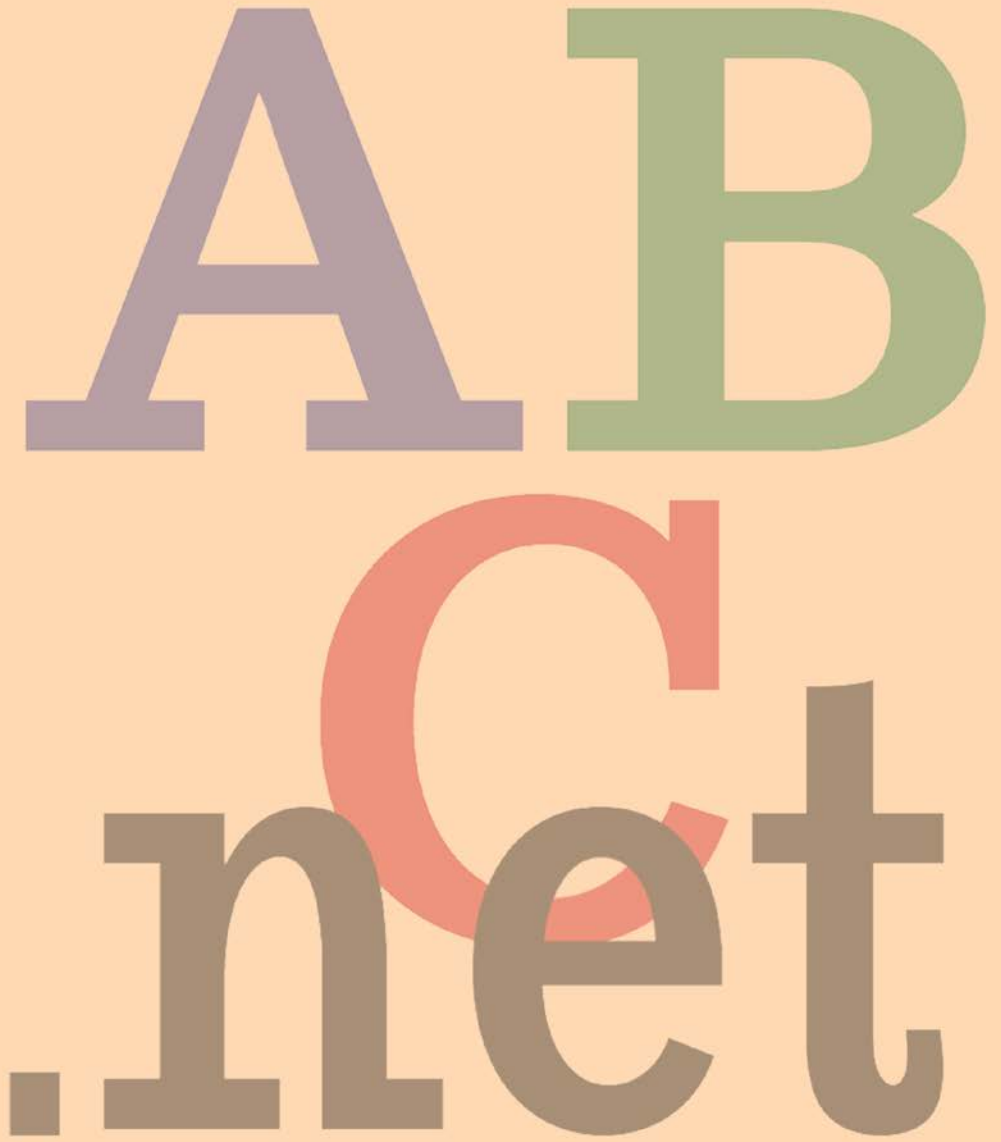
Александр Осипов

PascalABC.NET: выбор школьника
Часть 1



Александр Осипов

PascalABC.NET: выбор школьника
Часть 1

Почему PascalABC.NET?
Базовые знания
Линейные алгоритмы
Алгоритмы с ветвлением
Циклические алгоритмы
Pascal в КИМ ЕГЭ и ОГЭ

70 типичных задач:
постановка, математическая модель,
программный код

Ростов-на-Дону
2020

УДК 004.432
ББК 32.973
О741

Рецензенты:

кандидат физико-математических наук,
доцент кафедры алгебры и дискретной математики
Южного федерального университета
С. С. Михалкович

кандидат физико-математических наук,
доцент, заведующий кафедрой информатики, физики и МПИФ
Оренбургского государственного педагогического университета
В. О. Дженжер

Осипов А. В.

О741 PascalABC.NET: выбор школьника. Часть 1. — 2-е изд., испр. и доп., /А. В. Осипов. – Ростов-на-Дону; Таганрог : Издательство Южного федерального университета, 2020. – 148 с.

Целевая аудитория книги – школьники и учащиеся иных общеобразовательных учреждений среднего образования. Книга может быть также полезна студентам младших курсов, учителям и преподавателям, интересующимся решением задач в современной версии языка Pascal. Приводится теория и дается решение задач по программированию из школьного курса информатики с максимальным использованием возможностей PascalABC.NET. Подборка задач позволяет использовать книгу в качестве ныне популярного «решебника», но главная цель – научиться писать современный короткий, понятный и эффективный код.

УДК 004.432
ББК 32.973

Оглавление

Оглавление ... 3
Предисловие разработчика PascalABC.NET ... 7
Предисловие рецензента ... 9
От автора ... 11
Глава 1. Базовые знания ... 13
1.1. Почему именно PascalABC.NET? ... 14
1.2. Основная программа и ее запись ... 19
1.3. Простейший вывод данных ... 22
1.3.1. Вывод текста ... 22
1.3.2. Вывод числовых данных ... 24
1.4. Основные числовые типы данных ... 25
1.4.1. Данные целого типа (integer) ... 26
1.4.2. Данные вещественного типа (real) ... 28
1.5. Построение арифметических выражений ... 30
1.5.1. Переменные ... 30
1.5.2. Арифметические операции ... 33
1.5.3. Приоритет арифметических операций ... 34
1.5.4. Стандартные функции ... 36
1.6. Оператор присваивания ... 37
1.7. Ввод данных ... 38
1.8. О ЕГЭ и ОГЭ ... 40
Глава 2. Линейные алгоритмы ... 43
2.1. Целочисленная арифметика ... 45
2.1.1. Перевод мер ... 45
2.1.2. Выделение цифр в числе заданной разрядности ... 48
2.2. Арифметика вещественных и целых чисел ... 49

2.2.1. Вычисления по известным формулам 50
2.2.2. Вычисления по формулам из геометрии 51
2.2.3. Прочие вычисления 54
Глава 3. Алгоритмы с ветвлением 57
3.1. Логический тип данных 58
3.2. Логические выражения 59
3.2.1. Операции отношения 59
3.2.2. Логические операции 60
3.2.3. О «короткой схеме» 61
3.3. Условный оператор 62
3.4. Условная операция 64
3.5. Алгоритмы без множественного выбора 66
3.5.1. Поиск минимумов и максимумов 66
3.5.2. Попадание точки в заданную область 67
3.5.3. Кусочно-заданные функции 70
3.5.4. Анализ цифр в числе заданной разрядности 72
3.5.5. Решение линейных и квадратных уравнений 75
3.5.6. Геометрия в вычислениях 79
3.6. Оператор выбора 82
3.7. Алгоритмы со множественным выбором 84
Глава 4. Циклические алгоритмы 87
4.1. Цикл с заданным числом повторений (loop) 88
4.2. Цикл с параметром (for) 88
4.3. Цикл с предусловием (while) 90
4.4. Цикл с постусловием (repeat) 90
4.5. Изменение хода выполнения цикла 91
4.5.1. Оператор break 91

4.5.2. Оператор continue ... 91
4.5.3. Оператор exit ... 91
4.6. Вложенные циклы ... 92
4.7. Задачи с использованием циклов ... 92
4.7.1. Ввод нескольких групп исходных данных ... 92
4.7.2. Ряды значений, вычисляемых по их позициям ... 95
4.7.3. Ряды значений, вычисляемых рекуррентно ... 99
4.7.4. Анализ цифр в произвольном целом числе ... 103
4.7.5. Табуляция функции одной переменной ... 106
4.7.6. Табуляция функции двух переменных ... 111
4.7.7. Нахождение НОД и НОК ... 113
4.7.8. Простые числа ... 115
4.7.9. Задача о сдаче («жадный» алгоритм) ... 117
4.8. Использование случайных чисел ... 119
4.9. Перевод между системами счисления ... 123
4.9.1. Перевод в десятичную систему ... 123
4.9.2. Перевод в систему по основанию N ... 124
4.10. Нахождение делителей числа ... 127
4.10.1. Нахождение всех делителей числа ... 127
4.10.2. Нахождение простых делителей числа ... 128
4.11. Решение задачи с сайта //acmp.ru ... 129
4.12. Вместо заключения ... 131
Приложения ... 133
П1. Прямоугольная система координат ... 134
П2. Уравнения плоских линий ... 135
П3. Геометрия в вычислениях ... 138
П4. Нахождение сумм и произведений ... 139

П5. Получение цифр в числе 141
П6. Схема Горнера для полинома 142
П7. Чтение программ в КИМ ЕГЭ и ОГЭ 144

Предисловие разработчика PascalABC.NET

Книга Осипова А.В. «PascalABC.NET: выбор школьника» описывает использование языка PascalABC.NET для решения задач базовой школьной программы. Это – первая книга с такой тематикой, опирающаяся исключительно на современные возможности нашего языка. Следует отметить, что одной из целей создания PascalABC.NET являлось именно обучение школьников современному программированию. Данная книга, несомненно, раскрывает это предназначение в полной мере.

В первую очередь, в книге показано, какие базовые средства языка следует использовать школьнику в повседневной жизни – при подготовке к урокам информатики и при сдаче ОГЭ и ЕГЭ. Дан также хороший обзор современных языков программирования, используемых для обучения школьников, обсуждаются их достоинства и недостатки. Хорошо обоснован выбор именно PascalABC.NET для обучения.

Первая часть – вводная, и на мой взгляд несколько переусложнена информацией о внутреннем представлении типов, их явных приведениях и шестнадцатеричном представлении. Для младших школьников такой материал – непосильный, и они будут его попросту пропускать. Поэтому я бы рекомендовал читать материал главы 1 «по диагонали» и сразу переходить к самому интересному – главам 2-4.

Начиная с главы 2, интерес к прочтению сильно возрастает – приведено большое количество небольших задач, решенных в современном стиле PascalABC.NET. Это – ровно тот стиль написания программ, который мы рекомендуем как разработчики языка. Главы 2-4 читаются легко, разнообразие материала несомненно увлечёт юного читателя, а полученный им в результате прочтения и практического применения опыт программирования окажется поистине бесценным.

Небольшая критика содержания в главах 2-4 состоит в использовании устаревших конструкций языка вместо новых. Например, в тексте программ в основном используется устаревшая условная операция (var min := a < b : a ? b), написанная в С-стиле, вместо новой и более естественной для Паскаля (var min := if a < b then a else b).

Кроме того, вообще не используются диапазоны и естественная проверка на попадание в диапазон: if x in 2..5.

Из альтернативных методических подходов для выделения цифр целого числа я бы использовал последовательное выделение цифр с конца: var c1 := x mod 10; x := x div 10; var c2 := x mod 10; и т.д. Такой подход отличается бóльшей простотой и регулярностью и не использует сложные выражения вида x div 1000 mod 100.

Книга в целом – замечательная, даёт пищу для ума начинающему программисту. Книга может быть также использована школьными учителями для обновления методик преподавания информатике.

Руководитель проекта PascalABC.NET,
директор Воскресной компьютерной школы
при мехмате Южного федерального университета,
Михалкович Станислав Станиславович

Предисловие рецензента

В 1781 году И. Кант опубликовал одну из самых значимых своих работ, философский трактат «Критика чистого разума». Однако в 1783 году ему пришлось написать введение (пролегóмены) к своей книге. Кант объяснял это так: «Я опасаюсь, что о моей «Критике чистого разума» будут неправильно судить, потому что не поймут её, а не поймут её потому, что книгу, правда, перелистают, но не захотят её продумать». А. В. Осипов в 2019 г. выпустил свой более чем 500-страничный труд, посвящённый языку программирования PascalABC.NET и, видимо, по схожим причинам сразу принялся за более короткое введение в тему.

Новая работа заметно отличается от предыдущей.

Во-первых, конечно, объёмом. Здесь даётся относительно краткое введение в язык для человека, который, возможно, никогда не занимался программированием, но обладает некоторыми математическими познаниями и ясностью мышления. Многое осталось «за скобками» изложения, тем не менее, для первого знакомства с языком программирования сведений вполне достаточно.

Во-вторых, книга явно нацелена на школу. Её читателями должны стать ученики, которым нужен современный язык программирования для решения обычных учебных задач. В то же время те, кто готовится к сдаче ОГЭ и ЕГЭ, тоже могут найти полезным предлагаемый материал. Уверенности в этом придаёт то, что программы на новом Паскале получаются короткими, ёмкими и легко читаемыми. Все эти качества языка позволяют тратить меньше времени на написание программ (особенно это касается заданий ЕГЭ 24, 25 и 27) и не совершать обидных ошибок, либо легко находить и исправлять их. Следует напомнить, что в 2021 году предполагается смена формата ЕГЭ по информатике: он должен стать «компьютерным». В этом случае PascalABC.NET будет, безусловно, лучшим выбором школьника, сдающего этот экзамен.

В-третьих, существенной особенностью настоящей книги является большая «плотность» примеров. Изучить программирование без

анализа готового кода практически нереально. Поэтому в тексте приведено решение множества таких задач, которые часто вызывают трудности у начинающих. Стиль написания кода не самый современный, что объясняется общей направленностью книги на новичков в программировании, а также тем, что мы видим перед собой только первую часть, охватывающую самое начало языка. Но давайте подождём следующих томов!

Теория без практики мертва, а практика без теории слепа. А. В. Осипов смог найти хороший баланс между этими двумя крайностями, поэтому книга получилась интересной и поучительной. Мне хочется надеяться, что автору хватит сил на завершение этого нового труда. Пожелаем ему успехов. А лучшим подарком для него станет поколение школьников, вооружённых самым лучшим учебным языком программирования.

Дженжер В. О., к. ф.-м. н., доцент, заведующий кафедрой информатики, физики и МПИФ Оренбургского государственного педагогического университета

От автора

Не стоит изучать язык, который не меняет вашего представления о программировании.

Алан Перлис,
американский учёный в области информатики

7 октября 2019 года на официальном Интернет-сайте PascalABC.NET была опубликована книга «PascalABC.NET: Введение в современное программирование». Книга, содержащая практически полное на момент публикации описание замечательного языка программирования, одинаково хорошо приспособленного и для целей обучения, и для создания достаточно серьезных проектов. Около полугода – срок, недостаточный для того, чтобы говорить о успехе или неудаче технической книги объемом почти шестьсот страниц. Но первые отклики уже имеются.

Быстрее всего оказалась реакция школьников. «Да мне пять страниц прочитать проблема!». В нынешней реальности, когда школьник учит лишь то, что входит в материалы ЕГЭ, оставляя «за бортом» преобладающую часть предмета, это не удивляет. Разговариваю с пятиклассницей: – Много книг за лето прочитала? – Да, целых пять! И называет книги уровня «Собор парижской богоматери». Как мог это прочитать и понять одиннадцатилетний ребенок? Разгадка оказалась проста. Читались не сами книги, а их краткое содержание, специально изданное на двух-трех страничках. А остальное можно посмотреть на YouTube, который теперь заменяет детям телевизор, потому что там без рекламы и можно быстро пропускать.

Студенты – те почитывают. Выборочно. Потому что надо сдавать. От прочих категорий, пока отзывов нет, но книгу скачивают.

Как ни удивительно, молчат учителя. Казалось бы, они в первую очередь должны заинтересоваться мощным современным языком программирования, в то же время похожем на привычный им Паскаль. Но нет: молчат. Пугают почти шесть сотен страниц? Но ведь эта книга – она не учебник, который надо изучать от корки до корки. Она

скорее расширенный справочник с некоторым замахом на энциклопедию языка. Увы – пока молчат.

И я решил сделать еще один шаг навстречу потенциальным читателям. Написать своего рода «поваренную книгу», в которой все излагается коротко. Где рассматривается необходимая теория и разбираются типовые задания, предлагаемые школьникам и студентам. Откуда можно взять шаблон решения задачи и подогнать его под свои условия. Еще одна попытка привлечь внимание школьников и учителей. Время покажет, будет ли этот шаг правильным.

Планируется, что книга выйдет в трех частях. Первая часть книги ориентирована на версию 3.7. В работе я постоянно консультировался с руководителем проекта PascalABC.NET Станиславом Станиславовичем Михалковичем, что обеспечило технический надзор за излагаемым материалом. Вадим Олегович Дженжер, со своей стороны, позволил удалить из текста множество досадных опечаток и иных огрехов. С таким сильным и надежным тылом работать над книгой было достаточно комфортно и я хочу еще раз, публично, высказать этим двум людям огромную благодарность за их труд и терпение.

Выражаю благодарность Златопольскому Дмитрию Михайловичу за его отличный сборник задач (Златопольский Д.М. Сборник задач по программированию. —3-е изд., перераб. и доп. —СПб.: БХВ-Петербург, 2011.—304 с.: ил. —(ИиИКТ)) – именно этот сборник послужил источником или прототипом большинства задач в моей книге.

А.Осипов

Глава 1

Базовые знания

В этой главе…

Почему именно PascalABC.NET?
Основная программа и ее запись
Основные числовые типы данных
Арифметические выражения
Стандартные математические функции
Ввод и вывод данных
О ЕГЭ и ОГЭ

А теперь без грамоты
Пропадёшь,
Далеко без грамоты
Не уйдёшь.
С. Я. Маршак
«Кот и лодыри»

Я долго думал, как лучше расположить материал, чтобы обеспечить быстрое вхождение в PascalABC.NET. Термин «порог вхождения в язык» в программировании подразумевает промежуток времени, проходящий от момента начала изучения некоторого языка до написания на нем несложных программ. К универсальным алгоритмическим языкам с небольшим порогом вхождения относят, например, BASIC, Pascal и Python. Именно этим можно объяснить популярность упомянутых языков в школьной информатике.

На порог вхождения в язык влияет множество факторов. В первую очередь – его синтаксис и семантика. Синтаксис определяет набор правил для построения и записи языковых конструкций, а семантика – смысл и назначение каждой синтаксически правильной конструкции. Как язык программирования, PascalABC.NET обладает четким, простым и ясным синтаксисом. Разработчики языка затратили много сил, чтобы введение множества современных механизмов чрезмерно не усложнило синтаксис и по возможности сохранило стройную семантику языка.

1.1. Почему именно PascalABC.NET?

С 1985 года в девятых классах советских школ появился обязательный предмет «Основы информатики и вычислительной техники». В нем изучались, в частности, основы алгоритмизации и программирования. Школьникам предлагалось записывать алгоритмы на выдуманном русскоязычном алгоритмическом языке (РАЯ), доступа к вычислительной технике у них не было. Позднее на смену РАЯ пришли «настоящие» языки программирования – Бейсик и Паскаль, а также язык Ершол (автор – академик А.П. Ершов), разработанный на базе Алгол-60. Позднее Ершол был доработан и сейчас он известен под названием КуМир («Комплект учебных Миров»). Некоторое,

небольшое количество школ, выбрало для обучения информатике язык C++.

В настоящее время на компьютерах в школах, лицеях и колледжах можно встретить языки КуМир, BASIC (QBasic, Visual Basic), Pascal (Turbo/Borland Pascal, Borland Delphi, Free Pascal, Lazarus, Pascal ABC, PascalABC.NET), C++, C#, Python (версий 2 и 3), Java, JavaScript, Golan и некоторые другие. Изучение причин такого разнообразия не входит в круг вопросов, которые освещаются в данной книге. Отметим лишь, что КИМ (контрольные измерительные материалы) ЕГЭ по данным сайта ФИПИ (http://fipi.ru) в 2020 году будут включать задания, содержащие фрагменты программ на пяти языках: Бейсик, Алгоритмический язык (КуМир), Python, Паскаль и C++. Поэтому если планируется сдавать ЕГЭ, школьник либо должен помнить КуМир, который он изучал в шестом или седьмом классе и в дальнейшем им не пользовался, либо один из четырех перечисленных выше языков. Поэтому изучение C#, Java, Golan, а также экзотики допустимо только в качестве второго языка.

Вернемся к вопросу о пороге вхождения в язык.

У ***C++*** он непомерно велик, как бы ни пытались утверждать иное его горячие приверженцы. Аргументы типа «Зато на нем реально пишут программы, а кто возьмет на работу пишущего на бейсиках и паскалях?» я рассматривать не буду в силу их полной неуместности. После школы на работу в нормальное место не возьмут, даже если если выпускник будет знать двадцать языков. Нужно высшее образование. Потому что серьезное программирование требует знаний, которые даются только вузами. Но в вузе всем нужным языкам будут учить с начала. Для школы же важно вести преподавание информатики с использованием не «ныне модного» языка программирования, а такого, который будет легко воспринят учениками. Потому что учить нужно не языку программирования, а умению алгоритмизации задач. Язык – лишь средство записи алгоритма, поэтому предпочтение должно отдаваться языкам, позволяющим кратко и ясно записывать алгоритмы. Это дает возможность отказаться от блок-схем и сэкономить время, которого школьной информатике хронически не хватает. Школьники, которых учат языку («Питон - это круто, это мэйнстрим

сейчас!»), а не умению составлять алгоритмы, обычно испытывают серьезные трудности с программированием. Язык они выучили, а вот решение задачи записать на нем не могут: нет алгоритма – нечего и писать.

Рассматривать ***Бейсик***, как первый алгоритмический язык, я смысла не вижу. Он ушел со сцены, совершенно устарев.

Python (Па́йтон) в последние годы стал входить в моду. Это современный язык с низким порогом вхождения. По поводу того, может ли он быть первым языком для школьника, есть много высказываний как «за», так и «против». Приводить их тут – будет отдельная книга. Основная критика в школах: жесткая привязка записи программы к отступам, динамическая типизация, привязка имен к регистру символов, необходимость практически по любому поводу обращаться к библиотекам (и надо знать, что в какой содержится), отсутствие нормальных двумерных и большей размерности массивов. Кроме того, Python - интерпретируемый скриптовый язык, а любому интерпретатору присуща медлительность, иногда просто драматическая. Известны случаи, когда программа на Python считала часами, а на том же паскале, к примеру, завершалась за десятки секунд. Отсутствие описаний переменных на самом деле намного большее зло, чем это может показаться. Вот простейшая программа на Python3

```
a = 5
c = a/7
print(a,c,sep=' ')
```

Как думаете, есть ли в ней ошибки и если есть, то сколько и где? Увы, по виду не скажешь. Но запустим программу:

```
File "C:/Python/Python38-32/tmp.py", line 3, in <module>
    print(a, c, sep = ' ')
NameError: name 'c' is not defined
```

Итак, ошибка в последней строке – не определена переменная с именем *c* (если сумеете перевести на русский язык). Чтобы найти ошибку, программу надо запустить на компьютере. Но в чем же она? Python сообщает, что переменная не определена и это следует понимать так: потребовалось значение переменной, а оно ей еще не было

присвоено. Нет значения – нет типа, ведь типизация динамическая (вот и проявление зла!). Но как это – нет значения? Строкой выше эта переменная значение получила. Но… видимо все же не эта. Другая. Как и многие современные языки, Python позволяет использовать в именах символы национальных алфавитов. И вот как раз в этой программе одна из переменных «с» - это символ латиницы, а вторая – кириллицы. Вот только в какой строке какая?

Примерчик поинтереснее. А в этой программе есть ошибки? Какие и где?

```
a = int(input())  # Так в Python оформляется ввод данных
c = a/3
if c < 3:
    print(a,3*c,sep=' ')
    b = 12
else:
    print(a,c-1,sep=' ')
    b = 'abcd'
print(b/2)
```

Запускаем, вводим значение 5

```
==== RESTART: C:/Python/Python38-32/tmp.py ===
5
5 5.0
6.0
```

Уфф… отработала. Значит, ошибок нет? Но не спешите так, это же не какой-нибудь плебейский паскаль, это крутой Python! И ошибки на самом деле есть. Две. Одна в той ветке, которая по **else** и куда можно попасть, введя значение 9 и больше. Там переменная *a* набрана кириллицей. Но об этом пользователь программы узнает лишь тогда, когда будет предпринята попытка эту строку выполнить. А пока это просто мина, ждущая своего часа. Вторая ошибка немного похитрее. Переменной *b* в этой ветке присваивается значение строки, содержащей буквы. Последний оператор программы вычисляет и выводит значение *b/2*, считая что *b* должно быть числом. Программа отработала лишь потому, что она пошла по первой ветке, где *b* получило значение 12 и действительно стало числового типа.

Вот что творит динамическая типизация в кривых ручонках! А у какого новичка в программировании они сразу прямые? Так что на Python пусть пишут любители работать на минном поле. Остальным, считающим что квесты и челленджи не нужно смешивать с уроками информатики, Python выбирать для обучения алгоритмизации не следует. Как второй язык – отличное решение. В этой же ситуации компилятор Паскаль находит ошибку еще на стадии компиляции.

Ах да, может вы думаете, что извлечь квадратный корень из числа в Python так же просто, как в каком-нибудь КуМир или Паскаль? Радостно написать одну строчку print(Sqrt(2)), еще раз подумав: «а эти, со своими дурацкими паскалями все еще begin … end пишут», и увидеть результат? Спешу разочаровать: получите сразу две ошибки. Во-первых, Python не знает, что такое Sqrt. Функция извлечения квадратного корня хранится у него в библиотеке Math, имя которой надо указать перед функцией и отделить от нее точкой. Значит, print(Math.Sqrt(2)) ? Снова не угадали: библиотеку сначала надо подключить оператором import:

```
import Math
print(Math.Sqrt(2))
```

Но нет! Нет, пишет такого модуля. И правда, его нет. Потому что Python – язык регистрозависимый и библиотеку зовут на самом деле math.

```
import math
print(math.Sqrt(2))
```

Третий акт мерлезонского балета тоже оказывается неуспешным:

```
print(math.Sqrt(2))
AttributeError: module 'math' has no attribute 'Sqrt'
```

В переводе на русский язык: модуль math не имеет атрибута Sqrt. Не будем выяснять, что это за атрибут такой и почему его модуль не имеет. На самом деле, в библиотеке math не нашлось Sqrt. Потому что там – sqrt ! И что, вы сейчас подумали, если везде писать маленькие буквы, наступит полное счастье? Попытайтесь, конечно, если будете изучать Python. А что Паскаль? Все функции компилятор знает и так. Просто укажите нужную. Sqrt, sqrt, SQRT – как вам больше нравится.

Семейству языков ***Pascal*** в школьной информатике отведена особая роль. Прежде всего, паскаль – это почти как КуМир, только с английскими ключевыми словами. На него легко перейти. Далее, язык создавался именно для обучения и в этом качестве он превосходно был продуман. Он достаточно краток, ясен и нагляден, записи его операторов отлично укладываются на образ мышления. По этому языку выпущено большое количество книг, в том числе учебников для школы. Беда лишь в том, что Паскаль уже лет пятнадцать не развивался как язык и его концепции отражают прошлый век.

Проект ***PascalABC.NET*** призван дать языку Паскаль вторую жизнь, причем главным образом именно как языку для обучения программированию. Порог вхождения в язык низкий. На уровне школьных знаний PascalABC.NET практически совместим со всеми предшествующими диалектами языка Паскаль, что позволяет при необходимости запускать в его среде ранее написанные программы. Конструкции языка позволяют писать современный, короткий, наглядный и эффективный код. Язык компилируемый, со статической типизацией. Имеется интегрированная среда разработки и отладки, допускающая локализацию. В настоящее время в дистрибутив включена поддержка русского, английского и украинского языков. Язык мультипарадигменный. PascalABC.NET функционирует на платформе Microsoft .NET, используя её языковые возможности и библиотеки. Это делает его гибким, эффективным и постоянно развивающимся. Имеется возможность сочетать библиотеки, написанные на PascalABC.NET и других .NET-языках. На платформе Linux функционирует под Mono. Распространяется свободно по открытой лицензии LGPLv3. Официальный сайт http://pascalabc.net. Там же можно на бесплатной основе скачать литературу.

1.2. Основная программа и ее запись

В PascalABC.NET основная (главная) программа всегда представляет собой ***блок***. Блок – последовательность операторов (законченных фраз) языка, заключенная в операторные скобки **begin** … **end**. В основной программе после **end** всегда должна стоять точка, которая подсказывает нам: текст программы завершен.

```
begin
  оператор 1;
  оператор 2;  // а вот так пишут комментарии
  …
  оператор n;
 // и так тоже пишут комментарии,
 // когда они не помещаются в строке
end.
```

Блок в языке Паскаль имеет очень важное значение.

> Везде, где в соответствии с синтаксисом может быть указан оператор, на его месте может быть указан блок.

Операторы внутри блока принято втягивать вправо (для этого удобно использовать клавишу Tab), что делает структуру программы более наглядной, упрощая ее понимание. Принято – не значит обязательно: в PascalABC.NET, как и в подавляющем большинстве современных языков программирования, пробелы являются незначащими символами, любое их количество несущественно. Но лучше сразу выработать привычку делать форматирующие отступы. Помощь в этом может оказать интерфейс интегрированной среды разработки (IDE) PascalABC.NET, включающий кнопку для автоматического форматирования кода. В процессе ввода IDE отслеживает код и пытается делать отступы автоматически. Если это раздражает, автоформатирование всегда можно отключить.

Размещение конструкций языка на строке свободное: можно сколько угодно раз разрывать строку по любому пробелу, делая запись многострочной. Желательно при этом все же не проявлять фанатизма, пытаясь сделать программу похожей по начертанию на стихотворение Владимира Маяковского.

Я считаю свободную запись языковых конструкций естественной и правильной.

Антиподом здесь выступает язык Python, в котором пробелы являются важной частью синтаксиса, что сковывает руки программисту,

ставя правильность программы в зависимость от наличия или отсутствия пробела. Посмотрите код на языке Python3:

```
s = 0
k = 0
for i in range(1,20):
    if i % 3 == 0:
        s += i
        k += 1
print(s,k)
```

```
s = 0
k = 0
for i in range(1,20):
    if i % 3 == 0:
        s += i
    k += 1
print(s,k)
```

```
s = 0
k = 0
for i in range(1,20):
    if i % 3 == 0:
        s += i
k += 1
print(s,k)
```

В зависимости от величины отступа в строке k += 1 получаются разные результаты. Если же (например, при копировании) отступы будут утрачены, восстановление программного кода может оказаться непростой задачей.

```
s = 0
k = 0
for i in range(1,20):
if i % 3 == 0:
s += i
k += 1
print(s,k)
```

```
s = 0
k = 0
for i in range(1,20):
    if i % 3 == 0:
        s += i
      k += 1
print(s,k)
```

```
s = 0
k = 0
for i in range(1,20):
    if i % 3 == 0:
        s += i
          k += 1
print(s,k)
```

Во второй и третьей колонках сдвиг злосчастного оператора k += 1 в обе стороны относительно предыдущей строки на некоторое промежуточное число позиций является синтаксической ошибкой! Но вам повезло, в PascalABC.NET вы вольны расставлять операторы и их части в соответствии со своими желаниями.

Программа – блок, оканчивающийся точкой. Блок – последовательность операторов, которая начинается с **begin** и завершается **end**. Операторы внутри блока рекомендуется выделять отступами. Запись операторов свободная. После каждого оператора указывайте точку с запятой. В простых случаях, когда вся программа представляет собой небольшой единственный блок, можно вместо **begin** указать ##, а последнюю строку **end.** не писать вообще.

В первой части книги любая приведенная программа может начинаться с символов ##, либо быть окаймлена традиционным в языке Паскаль блоком **begin** … **end**.

1.3. Простейший вывод данных

PascalABC.NET обладает мощными развитыми средствами вывода и вы будете знакомиться с ними по мере изучения языка. Но пока достаточно и простейших возможностей.

В PascalABC.NET какие-либо «операторы» или «команды» вывода отсутствуют. Вывод осуществляется посредством программных единиц, именуемых процедурами и функциями. Первое время мы будем использовать две процедуры – Write и Print, а также их разновидности Writeln и Println.

Процедура Write выводит данные в строку, никак их не разделяя. Процедура Print после каждого выводимого данного делает пробел. В то же время, Write позволяет при желании определить формат вывода данных, а Print лишена этой возможности. Разновидности Writeln и Println по окончании вывода дополнительно осуществляют переход к следующей строке.

1.3.1. Вывод текста

Это самое простое. Все, что надо вывести, заключаем в апострофы. Например

```
Println('Этот текст будет выведен, как написано');
```

Конечно, тут можно было написать и Writeln. Если понадобится вывести знак апострофа, его в записи надо просто удвоить.

Вывод нескольких одинаковых символов лишь немного сложнее обычного вывода текста. Пусть требуется на длине 50 символов вывести по центру надпись «Лабораторная работа №3». Для таких случаев PascalABC.NET предлагает некоторые средства, упрощающие труд программиста, но пока сделаем это, используя наши скудные знания. Нам поможет обычная арифметика. Текст содержит 22 символа, следовательно на долю пробелов, заполняющих остаток строки, остается 50 – 22 = 28 символов. Это по 28 / 2 = 14 пробелов справа и слева. Слева эти 14 пробелов надо вывести, а справа в этом необходимости нет. Достаточно вывести заголовок и сменить строку. Окон-

чательно наша задача формулируется так: вывести 14 пробелов и после них заданный текст. Набирать эти 14 пробелов в тексте программы? Только не в PascalABC.NET!

```
## Writeln(14 * ' ', 'Лабораторная работа №3')
```

14 умножить на пробел? А что, есть возражения? Взять пробел 14 раз. Можно и наоборот написать, пробел умножить на 14. Так размножают и одинаковые фрагменты текста, содержащие больше одного символа. Повеселитесь, выведите тысячу звездочек. Или две. Задумайтесь о том, насколько беспросветно серой была ваша жизнь без PascalABC.NET. Как вариант, в данном случае можно было использовать форматный вывод «:36», но о форматах будет сказано позже.

А теперь нарисуем елочку. Как ни странно, учителя любят давать подобные задачи. И находятся ученики, не справляющиеся с этим! Решение этой задачи можно записать достаточно коротко, но до подобного кода вам пока что далеко. Так что просто посмотрите.

```
##
foreach var k in Seq(1, 3, 1, 3, 5, 7, 1, 3, 5, 7, 9, 11, 1, 1) do
  Writeln(k * '*':(8 + (k + 1) div 2));
```

А пока придется написать код в семь раз длиннее, зато он будет очень прост и понятен – только операторы, содержащие вывод.

```
##
Writeln(8 * ' ', '*');                          *
Writeln(7 * ' ', 3 * '*');                     ***
Writeln(8 * ' ', '*');                          *
Writeln(7 * ' ', 3 * '*');                     ***
Writeln(6 * ' ', 5 * '*');                    *****
Writeln(5 * ' ', 7 * '*');                   *******
Writeln(8 * ' ', '*');                          *
Writeln(7 * ' ', 3 * '*');                     ***
Writeln(6 * ' ', 5 * '*');                    *****
Writeln(5 * ' ', 7 * '*');                   *******
Writeln(4 * ' ', 9 * '*');                  *********
Writeln(3 * ' ', 11 * '*');                ***********
Writeln(8 * ' ', '*');                          *
Writeln(8 * ' ', '*');                          *
```

1.3.2. Вывод числовых данных

Чтобы вывести значение некоторого выражения, это выражение достаточно написать в списке элементов вывода процедур Write или Print. Оно будет вычислено и полученное значение выведено. Несколько выводимых значений отделяются друг от друга запятыми. Чередуя числовые данные и текст, можно оформлять вывод нужным образом

```
##
Writeln('Теперь наша программа может написать, что 2x2 = ', 2 * 2)
```

Получим на выводе фразу

```
Теперь наша программа может написать, что 2x2 = 4
```

Здесь два элемента вывода – текст и арифметическое выражение. Если что-то непонятно – про арифметические выражения сказано дальше.

В заключение – немного о форматах, которые можно указать при выводе чисел в процедуре Write.

Обратите внимание на вывод значения выражения 2 * 2, равного 4. Результат занял ровно одну позицию в строке вывода, поскольку большего числу 4 и не нужно. Но как организовать вывод нескольких чисел в столбик, если количество цифр в них неизвестно? Как выровнять числа с дробной частью по позиции десятичной точки? Эти проблемы решает формат.

Выводимое числовое значение занимает определенное место (***поле вывода***), в котором размещается определенное количество цифр (***позиций***). Если ширина поля вывода больше количества выводимых позиций, свободное место слева заполняется пробелами. Что произойдет, если число не вместится в поле вывода? Паскаль умный, он игнорирует формат и выведет число без потерь в знаках.

Формат вывода целых значений имеет вид :n, где n – ширина поля вывода. Для вещественных чисел используется формат :n:k, где k – количество позиций дробной части (если не указать k, Паскаль опре-

делит его сам). Дополнительный бонус – округление при выводе до k знаков после запятой. Сложившаяся практика предлагает писать для одиночных значений формат в виде :0:k – под целую часть нужное число позиций будет выделено автоматически.

```
##
var a := 45637.52;
Write(a, ' мм = ', a / 10:0:1, ' см = ', a / 1000:0:2, ' м')
```

```
45637.52 мм = 4563.8 см = 45.64 м
```

1.4. Основные числовые типы данных

PascalABC.NET позволяет работать с большим количеством числовых типов, но пока обойдемся лишь двумя из них, которые будем считать основными. Собственно, PascalABC.NET в большинстве случаев считает так же.

В арифметике числа бывают целыми и нецелыми (их называют в программировании вещественными). Целое число внешне отличается тем, что не имеет дробной части. Например, число 10 для нас целое. Как и число 4. Но если требуется разделить 10 на 4, то можно получить три разных результата в зависимости от назначения деления.

1. Если имеется 10 килограмм, сахара и надо развесить это количество поровну на 4 пакета, находим что 10 кг / 4 пакета = 2.5 кг/пакет. Мы получили нецелое число в результате деления двух целых чисел.
2. Если имеется 10 яблок и нужно раздать их поровну 4 детям так, чтобы каждый получил равное количество целых яблок, выполняем деление нацело с недостатком, получая 10 яблок / 4 человека = 2 яблока/человека. Здесь результат целый и еще имеется остаток в 10 – 2 × 4 = 2 яблока.
3. Если нужно испечь 10 пирогов, а в день можно испечь только 4, то понадобится 10 / 4 = 3 дня. Это целочисленное деление с избытком, потому что двух дней будет недостаточно.

Поучается, что в зависимости от ситуации мы решаем, как производить деление. Компьютер сам принимать решений не может, он лишь выполняет написанную нами программу. Поэтому нужно иметь

возможность указывать в программе как производить деление. Разные языки программирования позволяют делать это разными способами.

В Паскале имеется операция целочисленного деления с недостатком div, но ее можно применять только к данным целого типа. Также, есть операция деления /, дающая вещественный результат для числовых данных любого типа. А вот операции деления нацело с избытком нет (впрочем, ее нет и в других широко известных языках программирования).

Для PascalABC.NET пока примем, что целые числовые данные имеют тип **integer**, а вещественные – тип **real**.

Тип **real** является более общим по отношению к **integer**, поэтому если возникает необходимость, компилятор сам вставляет преобразование **integer** к **real**. Наоборот поступать запрещено! Компилятор не имеет права преобразовать **real** к **integer**, поскольку при этом можно утратить дробную часть.

```
(2 + 3) * 4 div 3 + 6 / 2
```

Здесь все числа будут иметь тип **integer**. Сначала вычисляется 2+3 = 5, затем 5*4 = 20, 20 div 3 = 6, а дальше… Дальше стоит операция деления. 6 / 2 = 3 с типом **real**. Теперь нужно сложить 6 типа **integer** с 3 типа **real**. Компилятор преобразует 6 к типу **real** и получается результат 6 + 3 = 9 с типом **real**. Вот так это работает.

Хотите проверить? Выполните такую программу:

```
## Print((2 + 3) * 4 div 3 + 6 / 2)
```

1.4.1. Данные целого типа (integer)

При записи значения предполагается, что если у него не указана дробная часть, значение имеет тип **integer**. Например, тип **integer** будут иметь числа 0, 15, -345, 1241, -12345678 и т.д. Для хранения таких чисел PascalABC.NET отводит 4 байта, поэтому числа должны укладываться в интервал $[-2^{31};2^{31})$. Этому интервалу удовлетворяют значения от -2147483648 до 2147483647. Запомните примерное

значение **2,1 миллиарда** и ориентируйтесь на него при оценке. Что произойдет, если выйти за указанный диапазон значений? Будет нехорошо. Очень нехорошо. Нехорошо – потому что данные будут представлены неверно. Очень нехорошо – потому что PascalABC.NET сообщит об этом не всегда. Самая большая неприятность в работе с числами целого типа.

Если написать 1291 * 1291 * 1291, то при построении кода программы компилятор «заметит», что результат для типа **integer** слишком велик и выдаст сообщение об ошибке «Переполнение в арифметической операции». Программа не будет откомпилирована и запустить ее не удастся. Но давайте обманем компилятор.

```
##
var a := 1291;
Print(a * a * a)
```

Теперь компилятор «не видит» итогового выражения. Он определяет переменную *a* типа **integer**, заносит в нее значение 1291, а потом строит код для умножения. Поскольку *a* имеет тип **integer**, компилятор не видит причины изменять тип результата. Мина заложена. Она рванет, когда запустим программу. Запускаем. И вот оно: программа выдает значение -2143282125. Без особых проверок ясно, что значение неверно. Не может произведение трех положительных чисел быть отрицательным! Подобными примерами хорошо ставить на место зарвавшихся недоучек, которые сходив на пару уроков информатики и списав кривой программный код из Интернет, спешат объявить себя гипер-мега-супер-пупер программистами.

С целыми числами можно выполнять различные арифметические операции. Сложение (+), вычитание (-), умножение (*), целочисленное деление (**div**) и получение остатка от деления (**mod**) дают результат такого же типа **integer**. Можно изменять знак числа, указывая перед ним операцию «-» и она тоже дает результат типа **integer**. Операции деления (/) и возведения в степень (**) имеют тип **real**, так что вернемся к ним несколько позже. О возведении в квадрат позаботились особо: его выполняет функция с именем Sqr, вот только результат ее выполнения имеет другой целочисленный тип – **int64**,

поэтому если нужно возвести в квадрат некоторое k, лучше написать k * k и сохранить результату тип k.

PascalABC.NET позволяет также обращаться к достаточно большому набору функций, работающих с целыми числами. Но сначала рассмотрим другой числовой тип – вещественный.

1.4.2. Данные вещественного типа (real)

Как уже упоминалось, значения данных типа **real** могут содержать дробную часть. Такие данные PascalABC.NET хранит в восьми байтах. Диапазон представления данных от $-1.8 \cdot 10^{308}$ до $1.8 \cdot 10^{308}$, точность – 15 (иногда 16, но лучше на это не надеяться) цифр.

Вещественное число можно записывать с фиксированной точкой и с плавающей точкой. Запись с фиксированной точкой – известная из математики запись числа, в которой целая и дробная части разделены точкой (она играет роль привычной нам запятой). Например, 13.123, -0.0023, 1.15, 5.0. Обратите внимание на последнее число – фактически, оно целое, но тип **real** требует указывать дробную часть, даже если она нулевая. Поэтому 5 – это тип **integer**, а 5.0 – тип **real**.

С вещественными числами типа **real**, как и с целыми, можно выполнять различные арифметические операции. Сложение (+), вычитание (-), умножение (*), деление (/) и возведение в степень (**) дают результат такого же типа **real**. И функция Sqr для возведения своего аргумента в квадрат, возвращает результат типа **real**. Можно изменять знак числа, указывая перед ним операцию «-» и она тоже даст результат типа **real**.

И 13 / 5, и 13 / 5.0, и 13.0 / 5 – все это деление вещественных чисел. Нужно делить нацело? Пишите 13 **div** 5. А можно ли написать 13 **div** 5.0? Написать можно, но операция деления нацело определена в Паскале только для случая, когда по обе стороны от **div** указаны целочисленные значения, так что компилятор сообщит вам все, что думает о вашем умении программировать.

А вот Python поступит иначе. Он невозмутимо выполнит деление нацело для 13 // 5.0 и получит значение 2.0. Вещественное! Это открывает поистине «замечательные» перспективы! В самом деле, в Python типы данных в явном виде не описываются. Встретив в программе выражение *a // b*, сможете ли вы уверенно сказать, какого типа будет результат? Не сможете до тех пор, пока не определите типы *a* и *b* (или не узнаете, что тип *a* вещественный – float). В Python надо искать первое по времени присваивание значения этим переменным и определять тип из него. Но встретив при поиске оператор *a = c // x*, вы с радостью узнаете, что теперь надо разбираться с типами еще двух переменных. Все это делает разбор чужих программ на Python увлекательным квестом: вы сможете почувствовать себя специалистом по разгадыванию шифров.

Математический аналог представления числа с плавающей точкой вы тоже знаете: $1.3453 \cdot 10^{-6}$, $-3 \cdot 10^{14}$ и т.п. Такие числа записываются в программе почти так же, как в математике, только вместо 10 указывается латинская буква E (или e). Приведенные выше числа можно записать в виде 1.3453E-6 и -3e14.

Использование значений с фиксированной и плавающей точкой в программе равноправно: это просто различные формы изображения одной и той же величины. Но все же, значение 0.0000000000012345 лучше писать как 1.2345e-12 или 12.345e-13. Запись короче и читать удобнее.

Несколько слов о выводе вещественных значений посредством процедур Write и Print. Если формат вывода не указан, для значений с абсолютной величиной на интервале [0.001 ; 1.0), будет использован формат с фиксированной точкой, а на интервале (0.0 ; 0.001) – формат с плавающей точкой. Если формат указан, точка будет всегда фиксированной.

```
##
  Writeln(1.2e-5, 1.2e-5:12:7); // 1.2E-05   0.0000120
  Writeln(1.2e-2, 1.2e-2:12:7)  // 0.012    0.0120000
```

1.5. Построение арифметических выражений

В языках программирования под термином «выражение» понимают набор из имен переменных, констант, литералов, 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, если вещественным – соответственно о вещественном. Частными случаями выражения являются ***переменные*** и ***литералы***.

Числовой литерал (в переводе с латыни – буквальный) изображает значение числовой величины. Все числа, которые мы писали до сих пор, являются литералами. Например, в строке Println((2 + 3) * 4 div 3 + 6 / 2) литералами будут 2, 3, 4 и 6. Вы уже догадались, что текст в апострофах – это тоже литерал, только нечисловой?

1.5.1. Переменные

В выражение могут входить величины, значение которых разрешено изменять в процессе выполнения программы. Такие величины называются ***переменными***. Каждой переменной программист назначает имя (***идентификатор***), по которому затем обращается к ней. Тут все, как в алгебре. Каждая переменная имеет тип, который определяет тип хранимого в ней значения. Как мы уже договорились, пока будем использовать типы **integer** и **real**.

Перед тем, как первый раз использовать переменную, ее необходимо описать, дав возможность компилятору установить тип этой переменной. Ведь именно тип определяет, какой объем памяти нужно отвести для хранения значений переменной и какой код нужно строить при работе с этими значениями.

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;
```

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 его использовать, опустите указание типа в описании переменной, - и тип будет установлен автоматически по типу присваиваемого значения:

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

Примеры описаний переменных.

```
var a, b, gamma, w15: integer; // описание переменных списком
var bt: real; // описание одной переменной
var n: integer := 18; // описание, совмещенное с инициализацией
var s := 0.0; // описание с автовыведением типа real
var MyBytes := $C7; // шестнадцатиричное значение типа 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 разновидность описания переменных, объединенного с присваиванием начального значения и автовыведением типов.

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

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

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

Указывать тип переменных здесь нельзя – он определяется исключительно типом присваиваемого значения.

Конструкция вида

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

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

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

Использование неинициализированных переменных часто приводит к плохо обнаруживаемым ошибкам. В PascalABC.NET числовая переменная, которая при описании не получила начального

значения, обычно инициализируется нулем. Но полагаться на это можно только осознанно, поскольку инициализация гарантирована лишь для глобальных переменных из секции описания (мы пока о них не говорили). Хороший стиль программирования возлагает инициализацию переменных на программиста, а в требованиях ЕГЭ прямо указано, что за отсутствие явной инициализации переменных снимается один балл. Пока Вы программируете в среде .NET, она будет выполнять начальную инициализацию, но если попытаетесь перенести свой алгоритм за пределы .NET-языка, получите сюрприз.

1.5.2. Арифметические операции

Числовые литералы 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***.

К бинарным арифметическим операциям относятся сложение, вычитание, умножение, деление, возведение в степень, деление нацело и получение остатка 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. Рассмотрим пример.

```
##
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;
Print(a, b, c, d, e, f, g)
```

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 *e*. 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 все тот же: **integer**.

Если попытаться написать кортежное присваивание

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

результат окажется не тем, который ожидается. Здесь *d* зависит от *c*, но нет правила, указывающего что будет вычисляться раньше, *c* или *d*. Как и в обыкновенной операции присваивания, в кортежном сначала вычисляется все, что указано в правой части и лишь затем происходит присваивание значений переменным, записанным в левой части. Поэтому компилятор для выражения *c * a + 2 * b* зафиксирует ошибку и выдаст сообщение «Неизвестное имя 'c'».

На этой особенности кортежного присваивания основан обмен значениями двух переменных: *(a, b) := (b, a)*.

1.5.3. 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? Ведь если последовательно выполнить эти операции на калькуляторе, результат будет совсем иным. Дело в том, что мы еще в начальной школе усвоили правило арифметики: умножение делается раньше сложения. Чтобы не пу-

таться 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 приоритет, 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 – , и операция возведения в степень ** 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.

Пусть имеется выражение $\frac{a}{-b} - c + \frac{-d}{-e}$

В PascalABC.NET его можно записать как *a* / (-*b*) – *c* + (-*d*) / (-*e*), но можно и не мудрить со скобками, записав *a* / –*b* – *c* + –*d* / –*e*.

А как лучше? Лучше со скобками. В большинстве языков недопустимо писать два знака операции подряд, а вы не всегда будете пользоваться только PasacalABC.NET.

Ниже приведены примеры записи целочисленных арифметических выражений.

Математическая запись	Запись в PascalABC.NET	Ошибочная запись в 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))	

1.5.4. Стандартные функции

Арифметическое выражение может содержать вызовы ***функций***. Под функцией в программировании понимают некоторый самостоятельный фрагмент программы, имеющий имя и возвращающий результат. К функции можно обращаться из других мест программы (***вызывать*** функцию) путем упоминания ее 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.

Далее приведены некоторые из стандартных функций PascalABC.NET. При обозначении аргументов здесь принято, что аргументы m и n имеют тип **integer**, a и b – тип **real**, x и y – **integer** или **real**.

Abs(x) – абсолютное значение аргумента x
Max(x, y, …) – максимальное из значений x, y, …
Min(x, y, …) – минимальное из значений x, y, …
Random – случайное число типа **real** из интервала [0 ; 1)
Random(m) – случайное число из интервала [0 ; m-1]

Random(a) – случайное число из интервала [0 ; a)
Random(m, n) – случайное число из интервала [m ; n]
Random(a, b) – случайное число из интервала [a ; b)
Random2(m) – кортеж из двух случайных чисел в интервале [0 ; m-1];
Random2(a) – кортеж из двух случайных чисел в интервале [0 ; a);
Random2(m, n) – кортеж из двух случайных чисел в интервале [m ; n];
Random2(a, b) – кортеж из двух случайных чисел в интервале [a ; b);
Random3(m) – кортеж из трех случайных чисел в интервале [0 ; m-1];
Random3(a, b) – кортеж из трех случайных чисел в интервале [a ; b);
Sign(x) – -1 при x < 0, 0 при x = 0 и 1 при x > 0;
Sin(x) – sin(x) с типом **real**;
Sqr(a) – a^2;
Sqr(m) – m^2 с типом **int64**. Не пугайтесь, он совместим с типом **integer**;
Sqrt(x) – $\sqrt{x}$ с типом **real**;
Trunc(a) – целая часть значения a с типом **integer**.

Полный перечень математических функций можно найти в Справке PascalABC.NET, доступной через меню «Справка» - «Системный модуль PABCSystem» - «Математические подпрограммы». В частности, кроме уже перечисленных, имеются функции ArcCos, ArcSin, ArcTan, Ceil, Cos, Cosh, DegToRad, Exp, Floor, Frac, Int, Ln, Log, Log10, Log2, LogN, Odd, Power, RadToDeg, Round, Sinh, Tan, Tanh.

1.6. Оператор присваивания

Оператор присваивания получается, если убрать из описания переменной с автовыведением типа служебное слово **var**. Этот оператор используется для ранее описанных переменных и позволяет присвоить им некоторое значение.

```
Имя переменной := выражение;
```

Имеется также ***кортежное присваивание***. Список в скобках – это есть и кортеж, но это материал второй части книги.

```
(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.

В PascalABC.NET имеются четыре оператора присваивания, в которых знак операции := видоизменен. Они взяты из языка C и в настоящее время имеются во многих языках программирования.

- *y += x* понимается, как «увеличить значение *y* на *x*» – аналог *y := y + x*;
- *y -= x* понимается, как «уменьшить значение *y* на *x*»;
- *y *= x* понимается, как «увеличить значение *y* в *x* раз»;
- *y /= x* понимается, как «уменьшить значение *y* в *x* раз».

Пока запомните, что тип **integer** автоматически приводим к типу **real** (но не наоборот!). Тип **int64**, который получается при вычислении квадрата целого числа посредством вызова функции Sqr, b и тип **integer** автоматически приводимы друг к другу. Но нужно понимать, что если после приведения **int64** к **integer** значение не разместится в **integer**, результат будет неверен.

1.7. Ввод данных

До сих пор мы писали программы, в которых все необходимые данные были известны заранее. Но типичная ситуация предполагает вычисления при задании значений некоторых данных. Например, мы хотим найти значение площади прямоугольника, длины сторон которого в процессе написания программы неизвестны. Здесь требуется операция, позволяющая ввести 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');
var ИмяПеременной := ReadReal('Текст приглашения ко вводу');
```

Если приглашение не нужно, оператор выглядит еще проще

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

Если переменная была описана ранее, ключевое слово **var** указывать нельзя поскольку 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 – до четырех переменных.

```
var (имя1, имя2) := ReadInteger2('Текст приглашения ко вводу');
var (имя1, имя2) := ReadReal2('Текст приглашения ко вводу');
```

Имеются также ReadInteger3, ReadReal3, ReadInteger4 и ReadReal4.

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

```
##
var (a, b) := ReadReal2('Введите длину и ширину прямоугольника:');
Print('Площадь прямоугольника равна', a * b)
```

Функция ReadReal2 (это именно функция) осуществляет прием с клавиатуры двух значений типа **real**, что подсказывает цифра 2 в ее названии, а кортежное присваивание помещает принятые значения в переменные *a* и *b* соответственно. Обратите внимание, что в процедуре вывода указано не имя переменной, а выражение. Это нормальная практика. Если значение выражения требуется в программе единственный раз, нет смысла создавать переменную и делать ей присваивание – такие действия лишь тратят память компьютера и увеличивают время работы программы.

Если нужно ввести значения данных разных типов или много значений, используется процедура Read со списком переменных известного типа.

```
##
var a, b, c: real;
var i, j: integer;
Read(a, b, c, i, j);
Print((3 * a + 5) / (b - i) + Abs(Sqrt(2 * j - c)))
```

Ниже – пример «маленького хулиганства», основанный на том, что функция возвращает результат обращения к ней. PascalABC.NET позволяет писать программы, не содержащие ни одной переменной!

```
## Print('Sпрямоугольника =', ReadReal('Длина:') * ReadReal('Ширина:'))
Длина: 21.7
Ширина: 15
Площадь прямоугольника: 325.5
```

1.8. О ЕГЭ и ОГЭ

В настоящее время КИМ ЕГЭ и ОГЭ содержат тексты и фрагменты программ, подлежащих анализу и написанных, в частности, на языке Turbo Pascal, т.е. самом раннем диалекте. Может возникнуть опасение, что изучая PascalABC.NET, школьник не сможет понять приводимого в заданиях кода. Эти опасения совершенно беспочвенны. Освоив PascalABC.NET, достаточно и получаса (!), чтобы научиться свободно понимать код Turbo Pascal. Писать программы на экзамене разрешено на любом языке при условии точного указания его наименования и версии. PascalABC.NET в силу своей мощности и лаконичности дает здесь неоспоримые преимущества. В разы короче код – в разы меньше шансов сделать ошибку или описку. Также, дополнительно высвобождается время на решение и проверку остальных заданий.

Язык PascalABC.NET полностью совместим с Turbo Pascal (а также, Free Pascal) в рамках КИМ, поэтому для тренировки и проверки знаний можно ввести программу так, как она приведена в задании или в школьном учебнике, запустить ее и посмотреть результат. Обратное неверно: программа, написанная в современном синтаксисе языка PascalABC.NET, не может быть выполнена ни в какой иной известной в настоящее время системе программирования. Отладчик в среде PascalABC.NET позволяет выполнять программу по шагам и/или приостанавливаться в любых ее точках с тем, чтобы контролировать значения переменных. Это делает работу по отладке программ комфортной и производительной.

В демонстрационной версии ФИПИ КИМ ЕГЭ 2021 программный код на языке Turbo Pascal приводится всего лишь в двух заданиях (6 и 22). От вас потребуется всего лишь **понять** этот код, а не писать программу.

В приложении П7 (и в приложении П1 второй части книги) изложены сведения, позволяющие легко добиться необходимого понимания. А писать вы можете на PascalABC.NET.

> Первая часть книги охватывает больше половины подмножества языка PascalABC.NET, необходимого для программирования школьных задач. Примеры решения задач с номерами 6 и 22 приведены в приложении П1 **второй** части книги, несмотря на то, что для решения достаточно знаний из первой части.

Материал первой части книги необходим для решения любых задач КИМ ЕГЭ, связанных с программированием. Он также достаточен для решения задач с номерами 16 и 17. Для решения задач с номерами 25, 26 и 27 потребуется знание материала второй части книги. Запланированная к написанию третья часть позволит решить задачу номер 24, связанную с обработкой символов. Также, в третьей части будет рассмотрена работа с файлами. Это связано с необходимостью читать файлы в задачах с номерами 24, 26 и 27.

Также хочу сообщить, что запланирована к написанию книга с рабочим названием «PascalABC.NET: ЕГЭ с комфортом», ориентированная на решение заданий КИМ ЕГЭ-2021. В ней будет рассмотрено решение задач, которые требуется выполнять посредством программирования на алгоритмическом языке. Будет также рассмотрено решение остальных задач КИМ, которые хотя бы в принципе имеет смысл программировать. В предоставленной ФИПИ демонстрационной версии КИМ в настоящий момент таких задач большинство.

Глава 2

Линейные алгоритмы

В этой главе…

Целочисленная арифметика
Перевод мер
Выделение цифр в числе
Арифметика вещественных и целых
Вычисления по известным формулам
Вычисления по формулам из геометрии

Раз дощечка, два дощечка – будет лесенка,
Раз словечко, два словечко – будет песенка...
М. Матусовский
«Вместе весело шагать...»

Линейный алгоритм («следование») – самый простой из всех алгоритмов. Он представляет собой последовательность блоков (участков) программы, которые однократно выполняются друг за другом. В то же время, это самый универсальный из алгоритмов. В самом деле, любая программа может быть укрупненно описана линейным алгоритмом из трех блоков:

- получение исходных данных;
- обработка данных;
- вывод результатов.

Каждый из этих трех блоков может иметь структуру произвольной сложности, но от этого он не перестанет быть блоком. Другое дело, что в современных диалоговых программах части этих блоков оказываются перемешаны между собой и линейность алгоритма внешне может быть не столь очевидна.

Любой алгоритм имеет некоторое начальное состояние, которое обеспечивается набором стартовых условий. Так, чтобы сложить «в столбик» два числа на бумаге, нужно иметь саму бумагу, карандаш или ручку, а также получить числа, которые требуется сложить. У компьютера проблемы с бумагой и карандашами отсутствуют, но числа ему все равно нужно предоставить. Для алгоритма нахождения суммы двух чисел сами эти числа будут называться *исходными данными.*

Исходные данные могут быть заданы непосредственно в алгоритме, либо переданы исполнителю в процессе выполнения алгоритма. Операция предоставления исполнителю исходных данных называется *вводом данных* или просто *вводом*. В тексте программы исходные данные могут быть указаны в операторах присваивания, а также получены при выполнении операций, реализующих ввод. Тип исходных данных определяется условиями задачи.

Под обработкой данных обычно понимается процесс преобразования исходных данных к результатам. Это наиболее емкая часть алгоритма, а большинство возникающих у школьника проблем связано именно с обработкой данных.

При выводе результатов нужно позаботиться о том, чтобы они представлялись в надлежащей форме, удобной для восприятия.

2.1. Целочисленная арифметика

Предполагается, что исходные данные и выводимые результаты имеют целочисленный тип (если это не олимпиадная задача, обычно достаточно типа **integer**). В простейшем случае могут быть заданы готовые формулы, по которым нужно произвести вычисление значения. В других случаях требуется самостоятельно составить алгоритм на основе имеющихся школьных знаний, часто не относящихся к информатике. Если формула задана или получается громоздкой, ее можно разбить на составные части, значения которых надлежит последовательно вычислить. Расстановка скобок, особенно при большой их вложенности, должна быть доведена у вас до автоматизма. Среда разработки и отладки PascalABC.NET облегчает отслеживание парности скобок (каждой открывающей скобке должна соответствовать закрывающая), подсвечивая текущую пару. При написании программы без компьютера не забудьте проконтролировать, чтобы количество открывающих скобок совпало с количеством закрывающих.

2.1.1. Перевод мер

Задача 2.1
Автомобиль проехал n километров. Найдите пройденный путь в метрах и сантиметрах.

Вспоминаем, что 1 км = 1000 м, 1 м = 100 см. Перевод из крупных мер в мелкие связан с умножением. Расстояние в метрах nm = 1000 × n, а расстояние в сантиметрах nsm = 100 × nm. Эти значения требуется вывести. Поскольку об оформлении вывода ничего не сказано, его можно сделать сообразно своему вкусу и умению. Значение n будем

задавать путем ввода, оформление которого также сделаем самостоятельно.

```
##
var n := ReadInteger('Путь в км:');
var nm := 1000 * n;
var nsm := 100 * nm;
Println('Путь в метрах равен', nm);
Print('Путь в сантиметрах равен', nsm)
```

```
Путь в км: 192
Путь в метрах равен 192000
Путь в сантиметрах равен 19200000
```

При оформлении ввода учитывайте, что ReadInteger и прочие функции ввода после подсказа автоматически вставляют пробел. При оформлении вывода не забывайте, что функция Print после каждого выводимого элемента автоматически вставляет пробел. Println по окончании вывода выполняет переход к следующей строке, Print такого перехода не выполняет, но в последней выводимой строчке нет нужды делать такой переход.

Решение этой же задачи можно записать и без промежуточных переменных, сократив себе работу.

```
##
var n := ReadInteger('Путь в км:');
Write('Путь в метрах равен ', 1000 * n, ', в сантиметрах - ', 100000 * n)
```

```
Путь в км: 192
Путь в метрах равен 192000, в сантиметрах - 19200000
```

Здесь для вывода использована процедура Write и все пробелы ставятся вручную. Если бы мы использовали функцию Print, между значением 192000 и запятой оказался пробел, а это некрасиво.

Задача 2.2

Дана масса в килограммах. Найти количество полных центнеров и полных тонн в ней.

Снова вспоминаем соотношение мер. 1 ц = 100 кг, 1 т = 1000 кг. Поскольку делается перевод в более крупные меры, потребуется выполнять деление. Указание о том, что нужны полные центнеры и тонны подсказывает, что деление должно быть целочисленным.

```
##
var m := ReadInteger('Задайте массу в килограммах:');
Write('Полных центнеров: ', m div 100, ', полных тонн: ', m div 1000)
```

```
Задайте массу в килограммах: 12397
Полных центнеров: 123, полных тонн: 12
```

Задача 2.3

С некоторого момента прошло 234 дня. Сколько полных недель прошло за этот период?

В полной неделе 7 дней. Следовательно, нужно найти результат целочисленного деления 234 на 7. Ввод в этой задаче не требуется.

```
## Print('В 234 днях полных недель -', 234 div 7)
```

```
В 234 днях полных недель – 33
```

Задача 2.4

Представить указанное число секунд, прошедших от ближайшей полуночи, в виде количества часов, минут и секунд. Результат вывести в формате ЧЧ:ММ:СС, незначащие нули не вставлять. 5 часов, 23 минуты и 9 секунд запишутся в виде 5:23:9.

В минуте 60 секунд, в часе 60 минут или 60 × 60 = 3600 секунд. Пусть s – исходное количество секунд. Тогда полное количество часов h будет равно результату целочисленного деления s на 3600, т.е. h := s **div** 3600. Остаток от этого деления (количество секунд, которые составляют неполный час) можно снова поместить в s посредством присваивания s := s **mod** 3600. Далее при помощи целочисленного деления находим полное количество минут: m := s **div** 60. Остаток от такого деления даст количество секунд в неполной минуте: s := s **mod** 60. При выводе значений h, m и s используем процедуру Write, обеспечивая требуемое оформление.

```
##
var s := ReadInteger('Количество секунд:');
var h := s div 3600;
s := s mod 3600;
var m := s div 60;
s := s mod 60;
Write('Показание часов ', h, ':', m, ':', s)
```

```
Количество секунд: 53051
Показание часов 14:44:11
```

2.1.2. Выделение цифр в числе заданной разрядности

Вам может быть полезен материал, приведенный в приложении П5.

Задача 2.5
В семизначном числе найти сумму второй и пятой слева цифр.

Вторая слева цифра – это 7 – 2 + 1 = 6-я справа. Пятая слева – это 7 – 5 +1 = 3-я справа.

```
##
var n := ReadInteger('Введите семизначное натуральное число');
Print('Сумма второй и пятой цифр равна',
    n mod 1000000 div 100000 + n mod 1000 div 100)
```

```
Введите семизначное натуральное число 4690129
Сумма второй и пятой цифр равна 7
```

Замечание. Если число может быть отрицательным, предварительно нужно найти его абсолютное значение, записав
var n := Abs(ReadInteger('...'));

Задача 2.6
В трехзначном числе переставить цифры в обратном порядке.

Обозначим цифры числа буквами s (сотни), d (десятки) и e (единицы). Исходное число имеет значение 100s + 10d + e. Требуется получить число 100e + 10d + s. Для чего сначала нужно найти s, d и e.

```
##
var n := ReadInteger('Введите трехзначное натуральное число');
var s := n div 100;
var d := n mod 100 div 10;
var e := n mod 10;
Print(100 * e + 10 * d + s)
```

```
Введите трехзначное натуральное число 825
528
```

Задача 2.7
Найти сумму и произведение цифр четырёхзначного числа.

Обозначим цифры числа буквами t (тысячи), s (сотни), d (десятки) и e (единицы). Найдем вначале каждую цифру, в потом их сумму и произведение.

```
##
var n := ReadInteger('Введите четырехзначное натуральное число');
var t := n div 1000;
var s := n mod 1000 div 100;
var d := n mod 100 div 10;
var e := n mod 10;
Write('Сумма цифр ', t + s + d + e, ', произведение ', t * s * d * e)
```

```
Введите четырехзначное натуральное число 3926
Сумма цифр 20, произведение 324
```

2.2. Арифметика вещественных и целых чисел

В задачах этой группы могут понадобиться следующие правила:

- если в бинарной операции хотя бы один из операндов типа **real**, результат будет также типа **real**;
- результат операции возведения в степень ** всегда имеет тип **real**;
- результат вызова функции Sqr(n) для аргумента n типа **integer** всегда **int64**. Типы **integer** и **int64** всегда приводимы один к другому;
- для приведения типа **real** к **integer** следует использовать или функцию Trunc(a), отбрасывающую дробную часть, или Round(a), выполняющую округление значения до целых;
- тригонометрические функции предполагают, что аргумент дается в радианной мере. Преобразовать градусную меру в радианную можно посредством функции DegToRad;
- обратные тригонометрические функции возвращают результат в радианах. Для перевода значения в градусы используйте функцию RadToDeg;
- числа π и e являются предопределенными в языке константами Pi и e;
- при делении числа на ноль или очень близкое к нему число может быть получено значение «бесконечность», в качестве которого выводится слово infinity либо символ ∞. При этом программа продолжает выполняться;
- при попытке выполнить недопустимую операцию (возведение отрицательного числа в степень, вычисление логарифма неполо-

жительного числа, вычисление арксинуса или арккосинуса от значения, превышающего 1 по абсолютной величине и т.п.) возвращается значение, выводимое как NaN (англ. Not a Number – не число) и программа продолжает выполняться.

2.2.1. Вычисления по известным формулам

Это простые задания для тренировки записи формул на языке программирования. Наиболее частая ошибка – неверное представление формул из-за игнорирования приоритета выполнения операций. Прочие ошибки – нарушение баланса открывающих и закрывающих скобок, неверные имена стандартных функций, пропуск знаков операций, неверная запись числовых литералов – обнаруживаются компилятором.

Задача 2.8
Вычислить значение функции $y = 4.5x^2 - 5x + 6$ при любом значении x, заданном вводом с клавиатуры.

Поскольку сказано, что x может принимать любые значения, а в функции есть коэффициент вещественного типа, логично объявить тип x также вещественным. Возведение в квадрат заменим умножением x на x для получения более короткой записи.

```
##
var x := ReadReal('x =');
var y := 4.5 * x * x - 5 * x + 6;
Print('y =', y)
```

```
x = -3.178
y = 67.338578
```

Задача 2.9
Составить программу вычисления значений функций при любых значениях x и y.

$$z = \frac{x + \frac{2y}{x^2}}{y + \frac{1}{\sqrt{x^2 + 10}}}; \quad q = 2.8\sin x + |y|$$

Важно лишь правильно расставить скобки. Типы переменных – **real**.

```
##
var (x, y) := ReadReal2('Введите значения x и y:');
var z := (x + 2 * y / x ** 2) / (y + 1 / Sqrt(x * x + 10));
var q := 2.8 * Sin(x) + Abs(y);
Write('z = ', z, ', q = ', q)
```

```
Введите значения x и y: 2.63 -5
z = -0.248959039622826, q = 6.37078649911448
```

Напоминание: аргументы функции **всегда** заключаются в круглые скобки.

2.2.2. Вычисления по формулам из геометрии

Задача 2.10
Найти площадь кольца по заданным внешнему и внутреннему радиусам.

Площадь кольца вычисляется по известной формуле

$$S = \pi(R^2 - r^2),$$

где R – внешний радиус, r – внутренний радиус.

```
##
var (R1, r2) := ReadReal2('Введите внешний и внутренний радиусы:');
Print('S =', Pi * (Sqr(R1) - Sqr(r2)))
```

```
Введите внешний и внутренний радиусы: 120 105.3
S = 10404.6721253506
```

Задача 2.11
Заданы координаты двух точек на плоскости. Найти расстояние между точками с точностью до трех знаков после запятой.

Используем формулу П3.1 из приложении П3.

```
##
var (xA, yA) := ReadReal2('Введите координаты первой точки:');
var (xB, yB) := ReadReal2('Введите координаты второй точки:');
var L := Sqrt(Sqr(xB - xA) + Sqr(yB - yA));
Write('Расстояние между точками составляет ', L:0:3)
```

```
Введите координаты первой точки: -12 3.6
Введите координаты второй точки: 7 -9.14
Расстояние между точками составляет 22.876
```

Задача 2.12

Треугольник задан координатами своих вершин. Гарантируется, что треугольник существует. Найти его периметр, площадь, а также радиусы вписанной и описанной окружностей. Вывод осуществлять с точностью 4 знака после запятой.

Обозначим координаты вершин ΔABC через $A(x_A, y_A)$, $B(x_B, y_B)$ и $C(x_C, y_C)$. Используя формулу П3.1, найдем длины каждой из сторон $a=L_{BC}$, $b=L_{AC}$, $c=L_{AB}$. Далее, найдем периметр P = a+b+c, полупериметр p = P/2 и площадь по формуле Герона:

$$S = \sqrt{p(p-a)(p-b)(p-c)}$$

Радиус вписанной окружности можно определить как r = S/p, описанной – как R = abc/(4S). Поскольку регистры букв в языке Паскаль не различаются, вместо p будем использовать p1, а вместо r – r1.

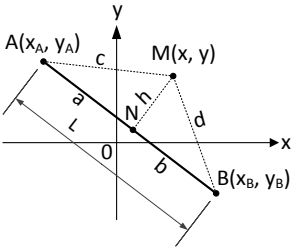
```
begin
  var (xA, yA) := ReadReal2('Введите координаты первой вершины:');
  var (xB, yB) := ReadReal2('Введите координаты второй вершины:');
  var (xC, yC) := ReadReal2('Введите координаты третьей вершины:');
  var a := Sqrt(Sqr(xC - xB) + Sqr(yC - yB));
  var b := Sqrt(Sqr(xC - xA) + Sqr(yC - yA));
  var c := Sqrt(Sqr(xB - xA) + Sqr(yB - yA));
  var P := a + b + c;
  var p1 := P / 2;
  var S :=Sqrt(p * (p - a) * (p - b) * (p - c));
  var R := a * b * c / (4 * S);
  var r1 := S / p1;
  Writeln('Периметр треугольника равен ', P:0:4, ', площадь ', S:0:4);
  Writeln('Радиус описанной окружности ', R:0:4);
  Write('Радиус вписанной окружности ', r1:0:4)
end.
```

```
Введите координаты первой вершины: -5 -3
Введите координаты второй вершины: 0 -6.1
Введите координаты третьей вершины: 11 5
Периметр треугольника равен 39.3988, площадь 821.7082
Радиус описанной окружности 0.5004
Радиус вписанной окружности 41.7124
```

Задача 2.13

Дана прямая, проходящая через заданные координатами точки $A(x_A, y_A)$ и $B(x_B, y_B)$. Определить кратчайшее расстояние от этой прямой до указанной точки M(x, y). Предполагается, что перпендикуляр, опущенный из точки M на отрезок AB, всегда попадает на него.

Из геометрии известно, что кратчайшим расстоянием в этом случае будет перпендикуляр, опущенный из точки M на прямую AB. Пусть он пересекает прямую в точке N. Соединив точку M с точкой A, получаем прямоугольный треугольник AMN. Обозначим его катеты буквами a и h, гипотенузу – буквой c. Аналогичным образом строим треугольник BNM.



Зная координаты точек A, M и B, можно найти длины отрезков c, d и L по формуле П3.1. Далее, приравняем площади треугольника AMB, полученные по формуле Герона и как половину произведения основания AB на высоту h, откуда найдем h.

$$\sqrt{p(p-c)(p-d)(p-L)} = 0.5Lh \rightarrow h = \frac{2\sqrt{p(p-c)(p-d)(p-L)}}{L};$$

$$p = 0.5(c+d+L);$$

$$c = \sqrt{(x-x_A)^2+(y-y_A)^2};\ \ d = \sqrt{(x-x_B)^2+(y-y_B)^2};\ \ L = \sqrt{(x_B-x_A)^2+(y_B-y_A)^2};$$

```
##
var (xA, yA) := ReadReal2('Координаты x и y 1-й точки:');
var (xB, yB) := ReadReal2('Координаты x и y 2-й точки:');
var (x, y) := ReadReal2('Координаты x и y точки M:');
var c := Sqrt(Sqr(x - xA) + Sqr(y - yA));
var d := Sqrt(Sqr(x - xB) + Sqr(y - yB));
var L := Sqrt(Sqr(xB - xA) + Sqr(yB - yA));
var p := 0.5 * (c + d + L);
var h := 2 * Sqrt(p * (p - c) * (p - d) * (p - L)) / L;
Print('Расстояние равно', h)
```

```
Координаты x и y 1-й точки: -5 3
Координаты x и y 2-й точки: 8 -4
Координаты x и y точки M: -2 -3
Расстояние равно 3.860527130428
```

Эту задачу можно решить намного быстрее и короче, без чертежей и выводов формул, если использовать готовую формулу П3.2, изменив ее сообразно условию

$$h = \left| \frac{(x_B - x_A)(y - y_A) - (y_B - y_A)(x - x_A)}{\sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}} \right|$$

```
##
var (xA, yA) := ReadReal2('Координаты x и y 1-й точки:');
var (xB, yB) := ReadReal2('Координаты x и y 2-й точки:');
var (x, y) := ReadReal2('Координаты x и y точки M:');
var h := Abs(((xB - xA) * (y - yA) - (yB - yA) * (x - xA)) /
    Sqrt(Sqr(xB - xA) + Sqr(yB - yA)));
Print('Расстояние равно', h)
```

2.2.3. Прочие вычисления

Задача 2.14

Известна стоимость 1 кг конфет, печенья и яблок. Найти стоимость покупки, если было куплено x кг конфет, y кг печения и z кг яблок.

Примитивная задача из арифметики для начальной школы. Обозначим стоимость 1 кг конфет через K, печенья – через P и яблок через B. Тогда стоимость покупки S составит K · x + P · y + B · z.

```
##
var (K, P, B) := ReadReal3('Стоимость 1 кг конфет, печенья и яблок:');
var (x, y, z) := ReadReal3('Куплено конфет, печенья и яблок (кг):');
Print('Сумма покупки составит', K * x + P * y + B * z)
```

```
Стоимость 1 кг конфет, печенья и яблок: 264 117 80
Куплено конфет, печенья и яблок (кг): 0.83 1.59 2.25
Сумма покупки составит 585.15
```

Задача 2.15

Даны шесть натуральных чисел a, b, c, d, e и f. Найти отношение минимального из средних арифметических каждой пары (a, b), (c, d), (e, f) к среднему геометрическому всех шести чисел.

Несмотря на то, что числа натуральные, их среднее геометрическое будет вещественным (средние арифметические также будут нецелыми в половине случаев). Среднее арифметическое пары чисел равно их полусумме, среднее геометрическое шести чисел равно корню шестой степени из их произведения.

$$m_2 = \frac{a_1 + a_2}{2};\quad g_6 = \sqrt[6]{a_1 \cdot a_2 \cdot \ldots \cdot a_6} = (a_1 \cdot a_2 \cdot \ldots \cdot a_6)^{1/6}$$

Минимум (как и максимум) в PascalABC.NET находится при помощи имеющейся в языке математической функции.

```
##
var a, b, c, d, e, f: integer;
Print('Введите 6 натуральных чисел:');
Read(a, b, c, d, e, f);
var mab := (a + b) / 2;
var mcd := (c + d) / 2;
var mef := (e + f) / 2;
var Делимое := Min(mab, mcd, mef); // вот он - минимум
var Делитель := (real(a) * b * c * d * e * f) ** (1 / 6);
Print('Результат:', Делимое / Делитель)
```

```
Введите 6 натуральных чисел: 126 142 98 100 104 113
Результат: 0.877611697589216
```

Обратите внимание: в произведении шести переменных первым сомножителем указана операция явного приведения типа **real**(a), которая принудительно приводит тип переменной a к **real**. Нельзя забывать, о верхней границе значений типа **integer**, примерно равной 2.1 миллиарда. Если перемножить шесть хотя бы двухзначных чисел, можно получить значение порядка $99^6 = 941\,480\,149\,401$, что примерно в 200 раз превышает эту границу. В результате будет получено неверное решение. Приведение одного из сомножителей к типу **real** вынудит компилятор построить код для работы с вещественными числами и угроза переполнения разрядной сетки будет устранена.

Глава 3

Алгоритмы с ветвлением

В этой главе…

Логический тип данных и логические выражения
О «короткой» схеме вычислений
Условный оператор и условная операция
Поиск минимумов и максимумов
Попадание точки в заданную область
Кусочно – заданные функции
Анализ цифр в числе заданной длины
Решение линейных и квадратных уравнений
Геометрия в вычислениях
Оператор выбора
Алгоритмы со множественным выбором

Будь у меня ветчина, я сделал бы себе яичницу с ветчиной. Будь при этом у меня яйца.

Кристофер Банч.
«Крылья урагана»

Программа – это разновидность записи алгоритма на некотором языке, которую компилятор умеет понимать с целью превращения программы в машинный код. Алгоритм строится на основе модели, с определенной степенью достоверности отображающей реальный мир. Но мир неоднозначен и постоянно заставляет делать какой-то выбор. Поэтому язык программирования должен включать средства для организации логических действий.

Необходимая гибкость в реализации логики достигается за счет введения в языке Паскаль данных логического (булевского) типа – **boolean**.

3.1. Логический тип данных

Как и для числовых типов, для логического типа можно описывать логические переменные, инициализировать их логическими значениями и использовать автовыведение типа. Логических констант в языке Паскаль две: **True** (истина) и **False** (ложь). Ими обычно и выполняют инициализацию. Но можно также инициализировать переменную результатом вычисления логического выражения. Если при описании переменная не получила начального значения, ей будет присвоено значение **False**, но писать программы, полагаясь на такую инициализацию – дурной вкус.

```
##
var a, b: boolean;
var c := True;
var (d, e, f) := (False, True, False);
Print(a, b, c, d, e, f)
```

```
False False True False True False
```

3.2. Логические выражения

Логическое выражение, подобно арифметическому, является неким аналогом математической формулы и результатом его вычисления будет одна из двух логических констант – **True** или **False**. Логические выражения могут строиться:
- на основе логического выражения с предшествующей ему унарной операцией логического отрицания;
- на основе двух арифметических выражений, связанных операцией отношения;
- на основе двух логических выражений, связанных логической операцией.

3.2.1. Операции отношения

Операции отношения хорошо известны из математики. Их шесть: равно « = », не равно « <> », меньше « < », меньше или равно « <= », больше « > » и больше или равно « >= ». Если с помощью одной из этих операций связать два арифметических выражения, мы получим ***утверждение***, которое будет истинным, либо ложным.

6 < 9.2 – истинное утверждение и значением логического выражения будет True.
3 = 5 – ложное утверждение; значением логического выражения будет False.
2 * 2 = 5 – ложное утверждение, False.
5 * 8 > (10 - 7) * 8 – True. 5 * 8 = 40, (10 – 7) * 8 = 24, 40 > 24 истинно.

Приоритет операций отношения ниже, чем у арифметических операций, что позволяет записывать утверждения без дополнительных скобок. Всегда сначала будут вычислены значения арифметических операций, а потом выполнится операция отношения.

3.2.2. Логические операции

В Паскале определена унарная логическая операция и три бинарных.

- унарная **not** –логическое отрицание («НЕ», инверсия);
- **or** – логическое сложение («ИЛИ», дизъюнкция);
- **and** – логическое умножение («И», конъюнкция);
- **xor** – сложение по модулю два, «исключающее ИЛИ»

Ниже приведена таблица истинности для перечисленных операций.

A	B	not A	A **and** B	A **or** B	A **xor** B
False	False	True	False	False	False
False	True	True	False	True	True
True	False	False	False	True	True
True	True	False	True	True	False

- функции, в том числе Sqr( ), имеют наивысший приоритет;
- унарные операции +, –, **not**, операция ** имеют приоритет 1;
- операции *, /, **div**, **mod**, **and** имеют приоритет 2;
- бинарные операции + и –, а также **or** и **xor**, имеют приоритет 3;
- операции отношения =, <>, >, >=, <, <= имеют приоритет 4.

Если требуется запрограммировать логическое выражение, содержащее логические операции, отсутствующие в языке Паскаль, нужно предварительно сделать эквивалентное преобразование.

Запись в формуле	Наименование операции	Запись в PascalABC.NET
A∨B, A+B, A|B	Логическое ИЛИ, дизъюнкция	A **or** B
A∧B, A&B, A · B, AB	Логическое И, конъюнкция	A **and** B
¬A, !A, $\bar{A}$	Логическое НЕ, отрицание, инверсия	**not** A
A⊕B, A+B(mod2)	Исключающее ИЛИ, сложение по mod 2	A **xor** B
A≡B, A↔B	Эквивалентность	**not** (A **xor** B)
A→B, A⊃B	Импликация	**not** A **or** B

Пусть требуется написать условие принадлежности точки с координатами (x, y) II или IV координатной четверти, исключая координатные оси.

Во II четверти абсциссы (x) точек отрицательны, а ординаты (y) положительны. В IV четверти наоборот, абсциссы положительны, а ординаты отрицательны. Абсциссы и ординаты объединяем по И.

Условие принадлежности точки II-й четверти: `(x < 0) and (y > 0)`.

Для IV-й четверти: `(x > 0) and (y < 0)`.

Объединяем эти условия по ИЛИ: (x < 0) **and** (y > 0) **or** (x > 0) **and** (y < 0).

В следующем примере требуется записать в языке Паскаль выражение $(x \vee y) \& z \equiv x \to \neg y + z \cdot x$

Здесь просто пользуемся таблицей, приведенной выше.

$$(x \vee y) \& z \equiv x \to \neg y + z \cdot x$$
$$\neg(((x + y) \& z) \oplus (x \to \neg y + z \cdot x))$$
$$\neg(((x + y) \& z) \oplus (\neg x + (\neg y + z \cdot x)))$$

```
not (((x or y) and z) xor (not x or not y or z and x))
```

3.2.3. О «короткой схеме»

Выражение P **or** Q истинно, если хотя бы P или Q истинно. Выражение P **and** Q ложно, если хотя бы P или Q ложно. Короткая схема вычисления значения логического выражения PascalABC.NET предполагает, что если значения P достаточно для решения вопроса об истинности выражения, значение Q не вычисляется.

Короткая схема может быть полезна, давая возможность не выполнять операцию, которая может привести к возникновению ошибки в программе.

3.3. Условный оператор

Условный оператор реализует выбор одной возможности из двух на основе анализа некоторого условия. Схематически он реализуется так:

```
ЕСЛИ УсловиеВыполняется ТО
   СделатьДействие1
ИНАЧЕ
   СделатьДействие2
```

В языке Паскаль вместо русских слова пишутся английские:

```
if P then
  A
else
  B;
```

Здесь P – это логическое выражение, значение которого проверяется. Если оно истинно (True), выполняется оператор A, если ложно (False) – оператор B. На месте оператора можно указать любой оператор языка, в том числе, другой условный оператор или блок. Чтобы было легче читать (и понимать) логику программы, в записи условного оператора применяются отступы.

Пусть даны натуральные числа a и b. Если их сумма четная, то вычислить произведение a×b, иначе вычислить утроенный остаток от целочисленного деления a на b.

```
##
var (a, b) := ReadInteger2('Введите два числа:');
if (a + b) mod 2 = 0 then  // можно (a + b).IsEven
   Print(a * b)
else
   Print(3 * (a mod b))
```

Вернемся еще раз к схематической записи условного оператора

```
if P then
  A
else
  B;
```

Если для P = **False** выполнять ничего не нужно, условный оператор можно сократить до записи вида

```
if P then
  A;
```

Если ничего не нужно выполнять для P = **True**, можно инвертировать P:

```
if not P then
  B;
```

Возможность записывать условные операторы в неполном формате (без **else**) может породить коллизию. Рассмотрим такую запись:

```
if P1 then
if P2 then A
else B;
```

В языке Паскаль существует простое правило: каждое **else** связывается с ближайшим предшествующим ему **then**, который не имеет собственного **else**. Согласно этому правилу приведенная выше последовательность операторов воспринимается как

```
if P1 then
  if P2 then A
  else B;
```

Чтобы реализовать иную схему вычислений, используем операторные скобки:

```
if P1 then
  begin
    if P2 then
      A
  end
else
  B;
```

3.4. Условная операция

Достаточно часто встречаются случаи, когда некоторое значение вычисляется в зависимости от выполнения определенного условия.

Пусть дана формула, определяющая функцию, заданную кусочно.

$$y = \begin{cases} 4x + 7, \text{ при } x < -5 \text{ или } x > 8 \\ 3x^2 - 11x + 6, \text{ при } -5 \le x \le 8 \end{cases}$$

Запишем программу с использованием условного оператора

```
##
var x := ReadInteger('x=');
var y: integer;
if (x >= -5) and (x <= 8) then
  y := 3 * x * x - 11 * x + 6
else
  y := 4 * x + 7;
Write('y=', y)
```

Что в этой программе не очень хорошо? Первое – объявляется переменная y и на это уходит целая строка. Надо заранее решить, какой тип будет иметь эта переменная. И второе, более неприятное – исходная формула распалась на два оператора.

Улучшить ситуацию поможет использование условной операции. Она пришла в PascalABC.NET из языков семейства C, где называлась ***тернарным выражением***. Условная операция по логике вычисления очень похожа на условный оператор и имеет формат

```
P ? A : B
```

Здесь P – некоторое логическое выражение, A и B – выражения (не операторы!), вычисление которых зависит от значения P. Если P = **True**, вычисляется значение выражения A, если P = **False**, вычисляется значение выражения B. Вычисленное значение становится значением условной операции.

Условная операция может быть также записана и в более привычном для языка Паскаль виде:

```
if P then A else B;
```

Условная операция может существенно сократить запись алгоритма. Приведенную выше программу с использованием условной операции можно записать следующим образом

```
##
var x := ReadInteger('x=');
var y := (x >= -5) and (x <= 8) ? 3 * x * x - 11 * x + 6 : 4 * x + 7;
Print('y =', y)
```

Если переменная y нужна только для вычисления и вывода результата, можно сэкономить еще одну строку программы и одну переменную.

```
##
var x := ReadInteger('x=');
Write('y=', (x >= -5) and (x <= 8) ? 3 * x * x - 11 * x + 6 : 4 * x + 7)
```

Условная операция может внутри себя (в выражениях) содержать другие условные операции и это позволяет реализовывать достаточно сложные вычисления. Кроме того, она сама может быть частью выражения.

Если в условной операции P ? A : B тип результатов A и B не совпадает, но оба типа могут быть автоматически приведены один к другому, делается попытка выбрать типом результата некоторый общий, в котором корректно отображаются оба типа. При невозможности корректного отображения будет выбран беззнаковый тип результата. Если один из типов невозможно автоматически преобразовать к другому, типом результата выбирается именно этот тип. Если автоматическое приведение ни одного из типов к другому невозможно, компилятор выдаст сообщение об ошибке.

Рассмотрим пример.

```
##
var (a, b) := (1, 2);
Print(if a > b then 1234567 else 152.408)
```

Литерал 1234567 имеет тип **integer**, литерал 152.408 – тип **real**. Тип **integer** приводится к **real**, но тип **real** не приводится к **integer**, поэтому выражнение получит тип **real**.

3.5. Алгоритмы без множественного выбора

3.5.1. Поиск минимумов и максимумов

Задача 3.1
Найти минимальное из двух произвольных чисел.

Если сказано, что числа произвольные, выбираем для них тип **real**. Алгоритм поиска минимума среди чисел a и b следующий: если a < b, минимум равен a, иначе он равен b.

```
##
var (a, b) := ReadReal2('Введите два числа:');
Print('Минимум равен', if a < b then a else b)
```

```
Введите два числа: 23.42 15.824
Минимум равен 15.824
```

Задача 3.2
Найти максимальное из трех целых чисел.

Сначала найдем максимальное из первой пары чисел, а затем сравним результат с третьим числом.

```
##
var (a, b, c) := ReadInteger3('Введите три целых числа;');
var m := if a > b then a else b;
m := if m > c then m else c;
Print('Максимум:', m)
```

```
Введите три целых числа; 4 7 2
Максимум: 7
```

Короткое альтернативное решение:

```
##
var (a, b, c) := ReadInteger3('Введите три целых числа;');
Print('Максимум:', Max(a, b, c))
```

Задача 3.3

Даны четыре натуральных числа. Найти процент отклонения значения максимального из чисел от среднего значения всех чисел с точностью до двух знаков после запятой.

```
##
var a, b, c, d: integer;
Write('Введите четыре целых числа: ');
Read(a, b, c, d);
var mx := Max(a, b, c, d);
var m := (a + b + c + d) / 4;
Write('Отклонение составляет ', (mx - m) / m * 100:0:2, '%')
```

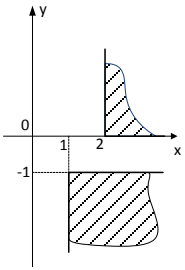
```
Введите четыре целых числа: 29 31 32 31
Отклонение составляет 4.07%
```

3.5.2. Попадание точки в заданную область

Задача 3.4

Если точка M(x, y) попадает в заштрихованную область или на ее границы, вывести «Да», в противном случае вывести «Нет».

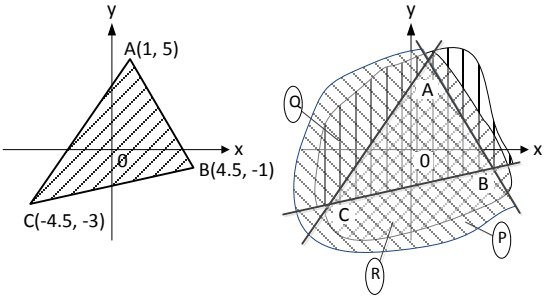
На рисунке даны две непересекающиеся области. Точка M может принадлежать одной из этих областей, либо не принадлежать ни одной из них. Следовательно, нужно записать условия попадания точки в каждую из двух областей и соединить их по «ИЛИ».



В верхней по рисунку области $x \geqslant 2$ и $y \geqslant 0$. В нижней области $x \geqslant 1$ и $y \leqslant -1$.

```
##
var (x, y) := ReadReal2('Введите координаты точки:');
if (x >= 2) and (y >= 0) or (x >= 1) and (y <= -1) then
  Print('Да')
else
  Print('Нет')
```

Задача 3.5
Если точка M(x, y) попадает в заштрихованную область или на ее границы, вывести «YES», в противном случае вывести «NO».



Треугольник, приведенный в левой части рисунка, можно представить пересечением трех областей. В правой части рисунка показаны область P, расположенная на линии AB и влево вниз от нее, область Q, расположенная на линии BC и влево вверх от нее, область R, расположенная на линии AC и вправо вниз от нее. Чтобы попасть в треугольник или на его границы, точка M должна принадлежать одновременно всем трем областям. Следовательно, нужно записать условия попадания точки в каждую из областей и соединить их по «И».

Границами перечисленных областей являются прямые линии, у которых известны координаты двух точек. Коэффициенты уравнения прямой, проходящей через эти точки найдем по формулам П2.1.

Для прямой AB $x_1 = 1$, $x_2 = 4.5$, $y_1 = 5$, $y_2 = -1$ и получаем $k = -12/7$, $b = 47/7$.
Для прямой AC $x_1 = 1$, $x_2 = -4.5$, $y_1 = 5$, $y_2 = -3$ и получаем $k = 16/11$, $b = 39/11$.
Для прямой BC $x_1 = 4.5$, $x_2 = -4.5$, $y_1 = -1$, $y_2 = -3$ и получаем $k = 2/9$, $b = -2$.

Если область расположена выше прямой, то $y > kx + b$, если ниже, то $y < kx + b$. Зная это, легко получить нужные логические выражения.

Для области P: $y \leqslant (47 - 12 \cdot x) / 7$
Для области Q: $y \leqslant (16 \cdot x + 39) / 11$
Для области R: $y \geqslant 2 \cdot x / 9 - 2$

Треугольник объединяет все три области: P ∧ Q ∧ R. И только теперь пишем программу.

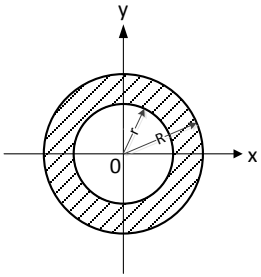
```
##
var (x, y) := ReadReal2('Введите координаты точки M:');
if (y <= (47 - 12 * x) / 7) and
   (y <= (16 * x + 39) / 11) and
   (y >= 2 * x / 9 - 2) then
  Print('YES')
else
  Print('NO')
```

```
Введите координаты точки M: 0.3 2.7
YES
```

В подобных задачах доля информатики невелика – все они решаются по одному шаблону. Поэтому школьники, которые жалуются на трудности с решением, на самом деле плохо знают математику.

Задача 3.6

Если точка M(x, y) принадлежит кольцу, образованному окружностями, большая из которых имеет радиус R, а меньшая – радиус r (не включая его границы), вычислить и вывести площадь этого кольца, в противном случае вывести текст «Точка не принадлежит кольцу».



Можно рассуждать следующим образом. Кольцо получается, если из круга радиуса R вырезать круг радиуса r. Тогда достаточно, чтобы точка M лежала внутри большей окружности и при этом вне меньшей окружности. Уравнение окружности известно, так что записываем два условия и связываем их по «И», поскольку выполняться они должны одновременно.

$(x^2 + y^2 < R^2) \wedge (x^2 + y^2 > r^2)$

Формула площади кольца также известна:

$S = \pi(R^2 - r^2)$

В программе обозначим R через r1, а r – через r2, поскольку Паскаль нечувствителен к регистру символов.

```
##
var (r1, r2) := ReadReal2('Наружный и внутренний радиусы кольца:');
var (x, y) := ReadReal2('Координаты x и y точки M:');
if (x * x + y * y < r1 * r1) and (x * x + y * y > r2 * r2) then
  Print('S =', Pi * (r1 * r1 - r2 * r2))
else
  Print('Точка не принадлежит кольцу')
```

```
Наружный и внутренний радиусы кольца: 11 8.4
Координаты x и y точки M: 5 -9
S = 158.461933447069
```

3.5.3. Кусочно-заданные функции

Задача 3.7

Дано вещественное x. Вычислить значение функции y(x), если

$$y = \begin{cases} 2 \text{ при } x > 2, \\ 0 \text{ при } 0 \le x \le 2, \\ 2x - 3.1 \text{ при } x < 0 \end{cases}$$

Прежде всего нужно проверить, покрывает ли область определения аргумента x весь возможный диапазон значений. Если покрывает, одно из условий можно будет не контролировать, поскольку оно выполнится по принципу исключения прочих. В данном случае область определения x распространяется на все вещественные числа, поскольку заданы интервалы (2;∞), [0;2], (-∞;0). Наиболее длинное описание y области [0;2]: оно содержит проверку обоих границ, поэтому это условие мы оставим для альтернативной ветки else.

```
##
var x := ReadReal('x =');
var y: real;
if x < 0 then
  y := 2 * x - 3.1
else if x > 2 then
  y := 2
else   // это и есть ветка, для которой истинно x ∈ [0;2]
  y := 0;
Print('y =', y)
```

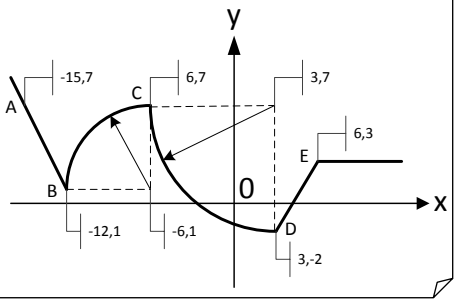
```
x = -0.17
y = -3.44
```

Это же решение можно написать намного короче, если использовать условную операцию.

```
##
var x := ReadReal('x =');
Print('y =', x < 0 ?  2 * x - 3.1 : x > 2 ? 2 : 0)
```

Задача 3.8

Дано вещественное x. Вычислить значение кусочно-заданной функции y(x), представленной на графике.



Функция определена для пяти участков изменения аргумента x: (-∞;-12], [-12;-6], [-6;3], [3;6] и [-6; ∞).

Поскольку функция не имеет разрывов, можно считать, что граничные точки принадлежат сразу обоим смежным участкам – это упростит программирование.

Уравнение прямой AB составим, используя формулу П2.1. Зная координаты точки A(-15; 7) и точки B(-12; 1), запишем уравнение прямой, проходящей через эти точки: y = -2x – 23.

На участке x ∈ [-12;-6] находится дуга BC, представляющая собой ¼ окружности с радиусом 6 и центром, расположенным в точке с координатами (-6;1). Уравнение такой окружности имеет вид $(x + 6)^2 + (y - 1)^2 = 36$. Решая уравнение относительно y получим

$$y = \pm\sqrt{36-(x+6)^2}+1$$

Дуга принадлежит верхней полуокружности, поэтому выбираем знак «плюс».

На участке x ∈ [-6;3] находится дуга CD, представляющая собой ¼ окружности с радиусом 9 и центром, расположенным в точке с координатами (3;7). Уравнение такой окружности имеет вид $(x - 3)^2 + (y - 7)^2 = 81$. Решая уравнение относительно y получим

$$y = \pm\sqrt{81-(x-3)^2} + 7$$

Дуга принадлежит нижней полуокружности, поэтому выбираем знак «минус».

На участке x ∈ [3;6] прямая DE проходит через точки D(3;-2) и E(6;3). Уравнение этой прямой будет иметь вид $y = 5 \cdot x / 3 - 7$.

И на последнем участке x ∈ [6; ∞] прямая имеет уравнение y=3.

Первый и последний участки описываются отношениями $x \leqslant -12$ и $x \geqslant 6$ соответственно. Остальные традиционно можно описать при помощи пары отношений, объединенных по «И». Так, для участка BC можно записать следующее выражение: $(x \geqslant -12)$ **and** $(x \leqslant -6)$. Кроме такого способа описания принадлежности значения x отрезку [a;b], в PascalABC.NET имеется конструкция вида x **in** a..b. Воспользуемся ей.

```
##
var x := ReadReal('x =');
Print('y =',
  if x <= -12 then
    -2 * x - 23
  else if x in -12..-6 then
    Sqrt(36 - Sqr((x + 6))) + 1
  else if x in -6..3 then
    -Sqrt(81 - Sqr((x - 3))) + 7
  else if x in 3..6 then
    5 * x / 3 - 7
  else 3)
```

```
x = -2.6
y = -0.0455659815234153
```

3.5.4. Анализ цифр в числе заданной разрядности

Задача 3.9

Если заданное трехзначное число читается одинаково слева направо и справа налево, вывести «Да». В противном случае вывести «Нет».

Число трехзначное, поэтому для выполнения приведенного в задании условия достаточно, чтобы первая цифра числа равнялась последней. Способ получения цифр числа приведен в приложении П5.

```
##
var n := Abs(ReadInteger('Введите трехзначное число:'));
Print(if n div 100 = n mod 10 then 'Да' else 'Нет')
```

```
Введите трехзначное число: 161
Да
```

Задача 3.10

Дано четырехзначное натуральное число. Определить:
а) входят ли в него цифра 2 или 7?
б) входят ли в него цифры 1 и 6?

Здесь достаточно много проверок, поэтому имеет смысл получить все четыре цифры числа, а затем приступить к проверкам.

Подобные задачи на практике решаются с помощью циклических алгоритмов и использования множеств, но поскольку этот материал еще не изучен, приходится обходиться тем, что есть.

Вначале получим все цифры числа – тысячи (t), сотни (s), десятки (d) и единицы (e). В данной задачи нет нужды в акценте на принадлежность цифры к конкретному разряду, так что имена выбраны исключительно для единообразия.

Условие «а» подразумевает, что для положительного ответа достаточно найти среди цифр числа лишь 2 или 7. В числе четыре цифры, каждая может совпасть с одной из двух искомых, так что придется сделать до 4 × 2 = 8 проверок, связанных по «ИЛИ». Тут нелишне вспомнить о «короткой схеме» (см. подраздел 3.2.3), использование которой позволяет компьютеру при истинности одного из отношений не выполнять лишние проверки.

Условие «б» для положительного ответа требует истинности одновременно двух отношений, что может породить достаточно сложную логику алгоритма. Поэтому введены две переменные типа **boolean**. Переменная d1 примет значение, указывающее на истинность высказывания о том, что среди цифр числа найдена цифра 1. Переменная

d6 позволит сделать аналогичное заключение о цифре 6. Тогда выражение d1 «И» d6 позволит сделать заключение о выполнении условия «б».

```
begin
  var n := ReadInteger('Введите натуральное 4-х значное число:');
  var t := n div 1000;
  var s := n mod 1000 div 100;
  var d := n mod 100 div 10;
  var e := n mod 10;
  if (t = 2) or (t = 7) or (s = 2) or (s = 7) or
      (d = 2) or (d = 7) or (e = 2) or (e = 7) then
    Println('Условие "а" выполняется')
  else
    Println('Условие "а" не выполняется');
  var d1 := (t = 1) or (s = 1) or (d = 1) or (e = 1);
  var d6 := (t = 6) or (s = 6) or (d = 6) or (e = 6);
  if d1 and d6 then
    Print('Условие "б" выполняется')
  else
    Print('Условие "б" не выполняется')
end.
```

```
Введите натуральное 4-х значное число: 5061
Условие "а" не выполняется
Условие "б" выполняется
```

Задача 3.11

Дано целое пятизначное число. Если куб суммы его цифр кратен натуральному числу b, вывести True, иначе вывести False.

Если число k кратно b, то k **mod** b = 0. В нашем случае k – куб суммы цифр числа n. Функция Abs позволит не думать о знаке числа n.

```
##
var n := Abs(ReadInteger('Введите пятизначное число:'));
var s := n div 10000 +
    n mod 10000 div 1000 +
    n mod 1000 div 100 +
    n mod 100 div 10 +
    n mod 10;
var b := ReadInteger('Введите число b:');
Print(s * s * s mod b = 0)
```

```
Введите пятизначное число: 53295
Введите число b: 7
False
```

Почему оператор для вычисления *s* занимает пять строк? Исключительно для наглядности. Если вам приятнее (и понятнее) запись вида

```
var s := n div 10000 + n mod 10000 div 1000 + n mod 1000 div 100 +
n mod 100 div 10 + n mod 10;
```

пишите в таком виде. Как и все другие языки программирования (кроме Python), Паскаль не стесняет вас в расположении и форматировании кода. Я выбрал многострочную запись, чтобы контролировать запись в «пирамидке нулей» при вычислении цифр числа.

3.5.5. Решение линейных и квадратных уравнений

Линейное уравнение имеет вид $ax + b = 0$, где $a \neq 0$. Его решением (корнем) является значение $x = -b / a$. В некоторых задачах, особенно олимпиадных, намеренно содержащих препятствия к получению корректного решения, значение $a = 0$ также считается допустимым, несмотря на то, что математически оно приводит к вырожденности линейного уравнения – об этом следует помнить.

Вычисление *x* производится при помощи операции деления «/», перед делением операнды приводятся к вещественному типу, так что лучше сразу определить для переменных *a* и *b* тип **real**. В PascalABC.NET при делении вещественного числа на ноль проблем не возникает (см. раздел 2.2), но ответ может оказать не в таком виде, в каком его требует задание. В этом случае нужно делать проверку либо до деления, либо проверять «на бесконечность» полученный результат при помощи логической функции **real**.IsInfinity, о которой вы пока ничего не знаете.

> ***Задача 3.12***
> *Написать программу для решения уравнения ax + b = 0. Значения a и b ввести с клавиатуры. Рассмотреть случаи, когда уравнение может иметь множество корней или не иметь корней вообще.*

Корень линейного уравнения определяется по формуле $x = -b / a$. При $a = 0$ возможны два случая. Если $b = 0$, исходное уравнение обращает-

ся в тождество $0 \equiv 0$ при любом x и корней бесчисленное множество. При $b \neq 0$ исходное уравнение принимает вид $b = 0$, что противоречит рассматриваемому значению, поэтому уравнение в этом случае корней не имеет. Для коэффициентов a и b выбираем тип **real**. Следует помнить, что если взять значение a достаточно близким к нулю, появится шанс получить «машинную бесконечность».

```
##
var (a, b) := ReadReal2('Введите коэффициенты a и b:');
if a = 0 then
  if b = 0 then
    Print('Корней множество')
  else
    Print('Корней нет')
else
  Print('x =', -b / a)
```

```
Введите коэффициенты a и b: 0.18 3.173
x = -17.6277777777778
```

Квадратное уравнение в общем виде записывается как $ax^2 + bx + c = 0$, где $a \neq 0$. Такие уравнение обычно решают при помощи вычисления вспомогательной величины – дискриминанта. Дискриминант определяется формулой $D = b^2 - 4ac$. Если дискриминант положительный, уравнение имеет два разных действительных корня. Если дискриминант равен нулю, оба действительных корня совпадают и для простоты говорят, что уравнение имеет один корень. Если дискриминант отрицательный, школьникам обычно говорят, что уравнение корней не имеет. На самом деле, уравнение не имеет лишь действительных корней. Как становится известно при дальнейшем изучении математики, в этом случае имеются два комплексно-сопряженных корня, но пока будем считать, что при $D < 0$ уравнение корней не имеет. Лица, продвинутые в математике, могут при вычислении дискриминанта явно указать тип **complex** и не тратить время на всякие анализы. Остальным следует использовать тип **real**.

Корни квадратного уравнения определяются по формуле

$$x_{1,2} = \frac{-b \pm \sqrt{D}}{2a}, \text{ где } D = b^2 - 4ac$$

Из приведенной формулы понятно, почему при $D = 0$ получаются два равных по значению корня. Также понятно, почему при $D < 0$ говорят,

что корней нет: возникает проблема с извлечением квадратного корня из отрицательного числа.

Задача 3.13

Найти корни квадратного уравнения $ax^2+bx+c=0$, если $a\neq0$. Коэффициенты a, b и c ввести с клавиатуры. Вывод осуществить с точностью 5 знаков после запятой. Если действительных корней нет, вывести соответствующее сообщение.

```
##
var (a, b, c) := ReadReal3('Введите коэффициенты a, b, c:');
var D := b * b - 4 * a * c;
if D > 0 then  // два разных корня
begin
  D := Sqrt(D);
  var x1 := (-b - D) / (2 * a);
  var x2 := (-b + D) / (2 * a);
  Write('x1 = ', x1:0:5, ',  x2 = ', x2:0:5)
end
else if D = 0 then
  Write('x = ', -b / (2 * a):0:5)
else
  Print('Корней нет')
```

```
Введите коэффициенты a, b, c: 3.8 -10 4
x1 = 0.49197,  x2 = 2.13960
```

Задача 3.14

Сумма квадратов двух последовательных натуральных чисел равна k. Найти эти числа.

Решение задачи состоит из двух этапов. На первом нужно построить математическую модель, выведя расчетные формулы. На втором – написать программу, реализующую эти формулы.

$n^2 + (n + 1)^2 = k$; $n^2 + n^2 + 2n + 1 - k = 0$; $2n^2 + 2n + (1 - k) = 0$;

Решаем полученное квадратное уравнение.

$D = 2^2 - 4 \times 2 \times (1 - k)$; $D = 4 - 8 + 8k$; $D = 4(2k - 1)$

Поскольку сумма квадратов натуральных чисел k также будет натуральным числом, дискриминант D всегда положительный и уравнение имеет два различных корня. Найдем их.

$$x_{1,2}=\frac{-2\pm\sqrt{4(2k-1)}}{2\times 2}=\frac{-1\pm\sqrt{2k-1}}{2}=0.5(-1\pm\sqrt{2k-1});$$

$x_1=0.5(-1-\sqrt{2k-1}$; $x_1<0$ - натуральное число не может быть отрицательным.

$x_2=0.5(-1+\sqrt{2k-1})$

Итак, задача свелась к нахождению значения x_2. Если оно будет целочисленным (в типе **real** дробная часть окажется равной нулю), решение имеется.

```
##
var k := ReadInteger('Введите натуральное число:');
var x := 0.5 * (-1 + Sqrt(2 * k - 1));
if Frac(x) = 0 then // при нулевой дробной части
  Write(k, ' = ', x, '^2 + ', x + 1, '^2')
else
  Write('Число непредставимо в искомом виде')
```

```
Введите натуральное число: 1201
1201 = 24^2 + 25^2
```

Задача только выглядела, как требующая решения квадратного уравнения, а на самом деле нужно лишь составить его и вывести расчетную формулу.

Задача 3.15

В прямоугольном листе жести со сторонами a см и b см (a > b) требуется вырезать прямоугольное отверстие площадью S см² так, чтобы его края располагались на одинаковом расстоянии m от краев листа. Определить расстояние m.

Прямоугольное отверстие будет иметь размеры a – 2m и b – 2m, тогда его площадь S составит (a-2m)·(b-2m).

$$S=(a-2m)(b-2m);\quad 4m^2-2(a+b)m+(ab-S)=0;$$

$$D=4(a+b)^2-4\times 4\times(ab-S)=4((a+b)^2-4(ab-S))=$$

$$4(a^2+2ab+b^2-4ab+4S)=4(a^2-2ab+b^2+4S)=4((a-b)^2+4S);$$

$$\sqrt{D}=2\sqrt{(a-b)^2+4S}$$

$$x_{1,2}=\frac{2(a+b)\pm 2\sqrt{(a-b)^2+4S}}{2\times 4}=0.25(a+b\pm\sqrt{(a-b)^2+4S})$$

Вот только теперь начинается информатика. И еще, нужно не забыть проверить полученные значения m на выполнение условий 2m < b и m > 0. Также, полезно установить контроль на вводимые данные, иначе площадь отверстия может превысить площадь листа.

```
##
var (a, b) := ReadReal2('Размеры сторон листа, см:');
var s := ReadReal('Площадь отверстия, кв.см:');
if s >= a * b then
  Print('Такое отверстие невозможно')
else
begin
  var d := Sqrt(Sqr(a - b) + 4 * s);
  var m := 0.25 * (a + b - d);
  Print('Расстояние:');
  if (m > 0) and (2 * m < b) then
    Print(m);
  m := 0.25 * (a + b + d);
  if (m > 0) and (2 * m < b) then
    Print(m)
end
```

```
Размеры сторон листа, см: 12 10
Площадь отверстия, кв.см: 25
Расстояние: 2.95049024320361
```

3.5.6. Геометрия в вычислениях

Сюда можно отнести задачи на вычисление элементов геометрических фигур и тел (сторон, площадей объемов и др.) и задачи на взаимное расположение геометрических объектов (например, нахождение кратчайшего расстояния между прямыми) или вопрос о попадании точки внутрь фигуры).

Задача 3.16
Даны три отрезка прямой длиной a, b и c. Можно ли из них составить треугольник?

Из геометрии известно условие существования треугольника: сумма длин двух любых его сторон должна быть больше длины третьей. Этим условием и нужно воспользоваться.

```
##
var (a, b, c) := ReadReal3('Введите длины сторон треугольника:');
if (a + b > c) and (a + c > b) and (b + c > a) then
  Print('Треугольник можно построить')
else
  Print('Треугольник построить нельзя')
```

```
Введите длины сторон треугольника: 15 8.64 11.9
Треугольник можно построить
```

Задача 3.17

Даны три отрезка прямой длиной a, b и c. Можно ли из них составить прямоугольный треугольник?

К условию существования треугольника из задачи 3.16 нужно добавить еще одно условие: в треугольнике должна выполняться теорема Пифагора. Квадрат большей из сторон (это будет гипотенуза) должен равняться сумме квадратов катетов (двух меньших сторон). Можно обеспечить нахождение в переменной *c* наибольшего из введенных значений. А можно каждой тройки значений набора {a, b, c} проверить выполнение теоремы Пифагора. Последний вариант записывается короче.

```
##
var (a, b, c) := ReadReal3('Введите длины сторон треугольника:');
if (a + b > c) and (a + c > b) and (b + c > a) then
  if (a * a + b * b = c * c) or (a * a + c * c = b * b)
      or (b * b + c * c = a * a) then
    Print('Можно построить прямоугольный треугольник')
  else
    Print('Можно построить непрямоугольный треугольник')
else
  Print('Треугольник построить нельзя')
```

```
Введите длины сторон треугольника: 15 20 25
Можно построить прямоугольный треугольник
```

Задача 3.18

Даны три отрезка прямой длиной a, b и c. Если из них можно составить треугольник, вычислить значения его углов с точностью до 0.1 градуса. Также вывести значение периметра треугольника P и его площади S.

Проверяем условие существования треугольника, как показано в задаче 3.16. Затем находим периметр треугольника и по формуле Герона определяем площадь.

$$P = a + b + c;\ \ p = \frac{P}{2};\ \ S = \sqrt{p(p-a)(p-b)(p-c)}$$

Существует очень полезная формула, связывающая площадь треугольника с двумя сторонами и углом между ними. Из нее легко определить угол. А затем из теоремы синусов получить остальные углы.

$$ab\sin C = 2S \rightarrow \sin C = \frac{2S}{ab},\ \ \mathrm{C} = \arcsin\frac{2S}{ab}$$

$$\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C} \rightarrow A = \arcsin\frac{a\sin C}{c},\ \ B = \arcsin\frac{b\sin C}{c}$$

Тригонометрические функции как в математике, так и в программировании обычно работают с радианной мерой. Для перехода от радианной меры к градусной в PascalABC.NET имеется функция RadToDeg.

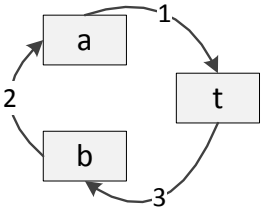
```
begin
  var (a, b, c) := ReadReal3('Введите длины сторон треугольника:');
  if (a + b > c) and (a + c > b) and (b + c > a) then
  begin
    var Per := a + b + c; // периметр
    var p := Per / 2;
    var S := Sqrt(p * (p - a) * (p - b) * (p - c)); // площадь
    var sinC := 2 * S / (a * b);
    var dA := RadToDeg(ArcSin(a * sinC / c));
    var dB := RadToDeg(ArcSin(b * sinC / c));
    var dC := 180 - da - db;
    Writeln('Периметр треугольника равен ', Per);
    Writeln('Площадь треугольника равна ', S);
    Write('Углы треугольника, град: A = ', dA:0:1,
        ', B = ', dB:0:1, ', C = ', dC:0:1)
  end
  else
    Print('Треугольник построить нельзя')
end.
```

```
Введите длины сторон треугольника: 18.2 25 21.343
Периметр треугольника равен 64.543
Площадь треугольника равна 189.964328204213
Углы треугольника, град: A = 45.4, B = 78.0, C = 56.6
```

Задача 3.19

Даны вещественные положительные числа a, b, c, x, y. Выяснить, пройдет ли кирпич с ребрами a, b, c в прямоугольное отверстие со сторонами x и y. Просовывать кирпич в отверстие разрешается только так, чтобы каждое из его ребер было параллельно или перпендикулярно каждой из сторон отверстия.

Пусть длины ребер кирпича расположены в порядке от меньшей к большей, т.е. a⩽b⩽c. Также, предположим что x⩽y. Тогда кирпич пройдет в отверстие, если a<x и при этом b<y. Осталось обеспечить нужный порядок следования размеров. Его мы добьемся, при необходимости обменивая значения переменных местами. Обмен можно выполнить посредством кортежного присваивания (a, b) := (b, a), либо используя процедуру Swap(a, b). Можно, конечно, использовать и примитивную процедуру обмена через промежуточную переменную *t*: *t := a; a := b; b := t*.




```
##
var (a, b, c) := ReadReal3('Размеры ребер кирпича a, b, c:');
var (x, y) := ReadReal2('Размеры отверстия x, y:');
if a > b then
  Swap(a,b);    // b >= a
if b > c then
  Swap(b, c);   // c >= b
if a > b then
  Swap(a,b);    // b >= a (если c был самым маленьким)
if x > y then
  Swap(x, y);
if (a < x) and (b < y) then
  Print('Пройдет')
else
  Print('Не пройдет')
```

```
Размеры ребер кирпича a, b, c: 60 25 40
Размеры отверстия x, y: 26 45
Пройдет
```

3.6. Оператор выбора

Условный оператор и условная операция позволяют выбрать один вариант из двух. Когда вариантов больше, приходится вкладывать

анализ одного условия в другое. При большом количестве вариантов получаются достаточно громоздкие и запутанные конструкции. Решение проблемы предлагает оператор выбора.

```
case Переключатель of
  СписокВыбора1: Оператор1;
  СписокВыбора2: Оператор2;
  …
  СписокВыбораN: ОператорN;
  else
    ГруппаОператоров
end;
```

Переключатель – это выражение так называемого **порядкового** типа. Нам важно сейчас, что оно может быть, в том числе, типа **integer**. Работа оператора выбора происходит следующим образом. Значение переключателя отыскивается в списках выбора, начиная от первого по порядку. Если поиск удачен, выполняется соответствующий оператор и на этом работа оператора выбора заканчивается. Если поиск неуспешен, выполняется группа операторов, указанная после **else**. Ветка **else** может отсутствовать и тогда при неуспешном поиске оператор вывода не делает ничего.

Рассмотрим пример. Достаточно часто встречается задача на сравнение двух целочисленных значений. Исходов у такого сравнения три: первое значение больше, значения равны, второе значение больше. Начинающие программировать обычно используют связку из пары условных операторов, но есть и другое решение – с помощью оператора выбора. Оно не совсем очевидно: ведь этот оператор использует константы порядкового типа, а сравнение – логическая операция. Но ведь ничто не мешает результат сравнения «вычислить», например, найти разность первого и второго значения. А переводить знак этой разности в число умеет функция Sign(x).

```
##
var (a, b) := ReadInteger2('Введите два числа:');
case Sign(a - b) of
  1: Print('Первое число больше');
  0: Println('Числа равны');
  -1: ('Второе число больше')
end
```

3.7. Алгоритмы со множественным выбором

Задача 3.20

Составить программу, которая в зависимости от порядкового номера дня недели (1, 2, ..., 7) выводит на экран его название (понедельник, вторник, ..., воскресенье).

Пример простейшей задачи, программируемой «в лоб», т.е. не раздумывая.

```
##
var n := ReadInteger('Укажите номер дня недели (1 - 7):');
case n of
  1: Print('Понедельник');
  2: Print('Вторник');
  3: Print('Среда');
  4: Print('Четверг');
  5: Print('Пятница');
  6: Print('Суббота');
  7: Print('Воскресенье');
else
  Print('Введен недопустимый номер дня')
end
```

```
Укажите номер дня недели (1 - 7): 3
Среда
```

Задача 3.21

Составить программу, которая в зависимости от порядкового номера месяца (1, 2, ..., 12) выводит на экран время года, к которому относится этот месяц.

Примеры записи списков выбора. Элементы списка перебираются по очереди и непринципиально, как строить список месяцев для зимы: 1, 2, 12 или 12, 1, 2. Элементы списка можно перечислять через запятую или задавать в виде диапазона (например, 3..5 означает «от трех до пяти», причем обе заданные границы также входят в диапазон).

```
##
var n := ReadInteger('Укажите номер месяца (1 - 12):');
case n of
  1, 2, 12: Print('Зима');
  3..5: Print('Весна');
  6..8: Print('Лето');
  9..11: Print('Осень');
else
  Print('Введен недопустимый номер месяца')
end
```

```
Укажите номер месяца (1 - 12): 2
Зима
```

Задача 3.22

Мастям игральных карт условно присвоены следующие порядковые номера: мастям «пики» - 1, «трефы» - 2, «бубны» - 3, «червы» - 4, а достоинству карт: «валету» - 11, «даме» - 12, «королю» - 13, «тузу» - 14 (порядковые номера карт остальных достоинств соответствуют их названиям: «шестерка», «девятка» и т.п.). По заданным номеру масти m (1-4) и номеру достоинства карты k(6-14) определить полное название (масть и достоинство) соответствующей карты в виде «Дама пик», «Шестерка бубен» и т.п.

```
##
var m := ReadInteger('Укажите код масти (1-4):');
var k := ReadInteger('Укажите достоинство карты (6-14):');
case k of
  6: Print('Шестерка');
  7: Print('Семерка');
  8: Print('Восьмерка');
  9: Print('Девятка');
  10: Print('Десятка');
  11: Print('Валет');
  12: Print('Дама');
  13: Print('Король');
  14: Print('Туз');
else
  Print('Достоинство карты указано неверно')
end;
```

```
      case m of
        1: Print('пик');
        2: Print('треф');
        3: Print('бубен');
        4: Print('червей');
      else
        Print('Масть карты указана неверно')
      end
```

```
Укажите код масти (1-4): 3
Укажите достоинство карты (6-14): 12
Дама бубен
```

Глава 4

Циклические алгоритмы

В этой главе…

Цикл с заданным числом повторений (loop)
Цикл с параметром (for)
Цикл с предусловием (while)
Цикл с постусловием (repeat)
Операторы break, continue и exit
Вложенные циклы
Ввод групп исходных данных
Ряды значений, вычисляемых по их позициям
Ряды значений, вычисляемых рекуррентно
Табуляция функций
Нахождение НОД и НОК
Простые числа
Жадный алгоритм (задача о сдаче)
Использование случайных чисел
Перевод между системами счисления

Это просто безумство: делать одно и то же раз за разом, и при этом ждать разных результатов.

Альберт Эйнштейн

С помощью алгоритмов, в которых шаги следуют строго друг за другом, возможно, временами разветвляя эти шаги, можно решить крайне ограниченный круг задач. Алгоритмы решения остальных требуют некоторого количества повторений своих отдельных частей. Такие повторяющиеся участки называют циклическими, а операторы языка Паскаль, реализующие соответствующие повторения – ***операторами цикла***. Цикл состоит из ***заголовка*** цикла и ***тела*** цикла. Заголовок определяет условие прекращения (или выполнения) цикла, а тело цикла содержит операторы, которые нужно повторять.

Язык PascalABC.NET предлагает пять разновидностей операторов цикла, четыре из которых будут рассмотрены здесь, а пятая (оператор **foreach**) – в следующей книге.

4.1. Цикл с заданным числом повторений (loop)

Это самый простой вид цикла. Его используют, когда число повторений тела цикла заранее известно, а значение номера прохода по циклу не используется. С помощью таких циклов часто оформляют вывод или строят элементы рисунков.

```
loop ЦелочисленноеВыражение do
    ТелоЦикла;
```

ЦелочисленноеВыражение вычисляется и его значение определяет количество повторений тела цикла. *ТелоЦикла* – любой оператор языка PascalABC.NET, либо блок.

4.2. Цикл с параметром (for)

Другое название цикла – ***цикл со счетчиком***. Он также организует повторение цикла фиксированное количество раз, но при этом автоматически увеличивает (или уменьшает) на единицу значение ***параметра цикла*** – некоторой переменной, указанной в заголовке

цикла. Пределы изменения параметра цикла задаются значениями *Выражение1* и *Выражение2*. Цикл **for** применяется, если в теле цикла используется значение параметра цикла.

```
for var ИмяПараметра := Выражение1 to Выражение2 do
    ТелоЦикла;
for var ИмяПараметра := Выражение1 downto Выражение2 do
    ТелоЦикла;
```

Выражение1 и *Выражение2* должны быть одного порядкового типа, совпадающего с типом параметра цикла или автоматически приводиться к нему. В противном случае компилятор зафиксирует ошибку. Пока примем, что тип всегда будет integer.

Рассмотрим работу оператора цикла с **to**. Перед первым выполнением тела цикла, параметр цикла получает значение *Выражения1*. Вычисляется значение *Выражения2*. Если оно не меньше значения *Выражения1*, выполняется тело цикла. Затем параметр цикла увеличивается на единицу (делается так называемый ***шаг***). Полученное значение параметра цикла вновь сравнивается со значением *Выражения2*. Как только значение параметра цикла станет больше значения *Выражения2*, цикл завершит свое выполнение.

Выражения, определяющие начальное и конечное значение параметра цикла, вычисляются один раз - перед первым входом в цикл. Значение параметра цикла в теле цикла программисту менять запрещено – это ошибка, фиксируемая компилятором.

Цикл с **downto** отличается тем, что параметр цикла не увеличивается, а уменьшается на единицу. Поэтому тело цикла выполняется лишь пока значение параметра цикла больше или равно значению *Выражения2*.

Количество выполнений тела цикла k можно получить по несложной формуле: k = |Выражение2 – Выражение1| + 1

4.3. Цикл с предусловием (while)

Рассмотренные циклы предполагают известное или вычисляемое число повторений. В то же время, существенное количество алгоритмов основано на неизвестном количестве повторений своих циклических частей (количестве итераций). Такие алгоритмы предполагают, что цикл выполняется (или завершается) при достижении каких-либо условий.

Одна из разновидностей итеративного алгоритма – цикл с предусловием. В заголовке цикла находится некоторое логическое выражение и пока оно истинно, выполняется тело цикла («сначала подумаем, потом сделаем»). Цикл завершится, когда условие станет ложным

```
while ЛогическоеВыражение do
    ТелоЦикла;
```

В качестве *ТелаЦикла* может быть записан любой оператор языка или блок.

Цикл с предусловием можно использовать в математических итерационных алгоритмах для проведения вычислений с заданной точностью, при вводе данных, когда их количество заранее неизвестно, а условие завершения ввода определено некоторым введенным значением, при поиске среди каких-то данных нужного элемента и во многих других случаях.

4.4. Цикл с постусловием (repeat)

Этот цикл похож на цикл с предусловием. Отличие в том, что сначала выполняется тело цикла, а потом проверяется, не следует ли этот цикл завершить («сначала сделаем, а потом подумаем»).

```
repeat
  ТелоЦикла
until ЛогическоеВыражение;
```

Это единственный из операторов цикла, в котором *ТелоЦикла* может содержать несколько операторов и при этом не быть заключено в операторные скобки.

Цикл с постусловием при истинности *ЛогическогоВыражения* завершается.

Истинность логического выражения в заголовке оператора цикла `while` определяет условие *выполнения* цикла, а в `repeat` – условие его *завершения*. Наличие предусловия в цикле `while` может привести к тому, что тело цикла не выполнится ни разу. Постусловие в `repeat` – залог того, что тело цикла обязательно выполнится хотя бы один раз. Для бесконечного цикла `while` указываем `while True do`, для бесконечного `repeat` указываем `until False`.

4.5. Изменение хода выполнения цикла

В процессе выполнения тела цикла могут возникать обстоятельства, требующие изменить ход его обычного выполнения, например, досрочно завершить выполнение цикла или прекратить выполнять текущий проход по телу цикла и начать следующий.

4.5.1. Оператор break

Оператор вызывает немедленный выход из цикла. Используется, если обнаружена нецелесообразность дальнейшего выполнения тела цикла. Помните, что после выхода из цикла все переменные, определенные в теле цикла с помощью **var**, а также параметр цикла **for**, описанный в заголовке цикла, становятся недоступны.

4.5.2. Оператор continue

Оператор немедленно передает управление заголовку цикла. Оставшаяся часть тела цикла не выполняется. Если условие завершения цикла, определенное заголовком, не выполняется, начинается очередное выполнение тела цикла. На практике этот оператор нужен редко, поскольку легко и без ущерба для наглядности заменяется условным оператором, пропускающим при необходимости оставшиеся операторы в теле цикла путем «упрятывания» их под **else**.

4.5.3. Оператор exit

Единственное назначение – немедленно завершить работу той программной единицы, в которой он встретился. Бывает очень полезен.

Может быть использован в любом месте программы независимо от наличия в ней циклов.

4.6. Вложенные циклы

Поскольку в теле цикла может находиться любой оператор, ничто не мешает разместить там оператор цикла. Так появляются вложенные циклы. Они вкладываются друг в друга, как матрешки, ровно столько раз, сколько этого требует алгоритм. Количество вложений циклов друг в друга в программировании называют ***глубиной вложения***.

Операторы **break** и **continue** действуют только в пределах того цикла, в теле которого они записаны. Если требуется прервать выполнение не только внутреннего цикла, но и внешнего по отношению к нему, приходится идти на различные программистские уловки.

4.7. Задачи с использованием циклов

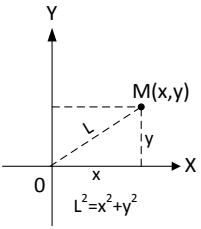
4.7.1. Ввод нескольких групп исходных данных

В этих задачах нужно получить результаты для нескольких различных исходных данных или нескольких групп данных. Алгоритм решения – цикл в котором очередная порция данных вводится с клавиатуры, затирая результаты предыдущего ввода.

Задача 4.1
На плоскости имеются n точек, заданных парой своих координат (x, y). Найти две точки, максимально удаленные от начала координат и одну минимально удаленную точку. Для каждой найденной точки вывести ее координаты и расстояние от начала координат.

Для поиска минимума и максимума понадобится выполнять сравнение. Но необязательно сравнивать расстояния L, можно сравнивать их квадраты. В качестве начального значения возьмем величины, которые в реальной задаче не встретятся. Например, для поиска максимума возьмем значение -1.0, которое заведомо меньше любого

L^2. Для поиска минимума возьмем наибольшую представимую в типе **real** величину **real**.MaxValue.



Требуется определить три точки, назовем их mx1, mx2 и mn. Для каждой точки надо хранить квадрат L и ее координаты, поэтому потребуется определить девять переменных. Так, для точки mx1 введем имена mx1L, mx1x и mx1y. Договоримся для определенности поддерживать соотношение mx1L ⩾ mx2L, для чего потребуются переприсваивания.

```
begin
  var n := ReadInteger('Количество точек (не менее 3):');
  if n < 3 then
    Print('Введено неверное значение')
  else
  begin
    var (mx1L, mx2L, mnL) := (-1.0, -1.0, real.MaxValue);
    var mx1x, mx1y, mx2x, mx2y, mnx, mny: real;
    loop n do
    begin
      var (x, y) := ReadReaL2('Задайте координаты x и y точки:');
      var L2 := x * x + y * y;
      if L2 > mx1L then
      begin
        (mx2x, mx2y, mx2L) := (mx1x, mx1y, mx1L);
        (mx1x, mx1y, mx1L) := (x, y, L2)
      end
      else if L2 > mx2L then
        (mx2x, mx2y, mx2L) := (x, y, L2);
      if L2 < mnL then
        (mnx, mny, mnL) := (x, y, L2)
    end;
    Writeln('Максимумы:');
    Writeln(Sqrt(mx1L), '(', mx1x, ', ', mx1y, ')');
    Writeln(Sqrt(mx2L), '(', mx2x, ', ', mx2y, ')');
    Writeln('Минимум:');
    Writeln(Sqrt(mnL), '(', mnx, ', ', mny, ')')
  end
end.
```

```
Количество точек (не менее 3): 5
Задайте координаты x и y точки: -5 -3.8
Задайте координаты x и y точки: 4 -2.1
Задайте координаты x и y точки: -5 -2
Задайте координаты x и y точки: 4 2.3
Задайте координаты x и y точки: 0 -4
Максимумы:
6.28012738724303(-5, -3.8)
5.3851648071345(-5, -2)
Минимум:
4(0, -4)
```

Задача 4.2

Факториалом натурального числа n! называется произведение всех членов натурального ряда чисел от 1 до n включительно. Найти и вывести значения 5!, 10!, 20! и 100!

Организуем цикл с количеством повторений, равным четырем. Для очередного введенного n вычисляется и выводится значение n!. Факториал получаем по алгоритму, приведённому в П4. С учетом вычисления 100! используем тип **real**.

```
##
loop 4 do  // 4 раза вводим значения n
begin
  var n := ReadInteger('n =');
  var p := 1.0;
  for var i := 1 to n do
    p *= i;
  Writeln(n, '! = ', p)
end
```

```
n = 5
5! = 120
n = 10
10! = 3628800
n = 20
20! = 2.43290200817664E+18
n = 100
100! = 9.33262154439441E+157
```

Обратите внимание, что и 20! не разместилось бы в типе **integer**.

4.7.2. Ряды значений, вычисляемых по их позициям

Категория задач, в которой обрабатывается ряд значений, каждое из которых однозначно вычисляется по своему порядковому номеру. Например, в ряду натуральных нечетных чисел, порядковый номер k определяет величину числа n по формуле n = 2k – 1, где k = 1, 2, … В случае, когда задана формула общего члена последовательности чисел, для известного количества чисел алгоритм обработки предусматривает использование цикла со счетчиком **for** с подстановкой значения параметра цикла в эту формулу. Если количество чисел заранее неизвестно и не может быть вычислено до входа в цикл, используются циклы с условием.

Задача 4.3
Найти сумму первых ста чисел натурального ряда.

Пример задачи, которую нужно решать на калькуляторе. Или на бумажке, «в столбик». Натуральный ряд чисел – арифметическая прогрессия, у которой первый член и разность равны единице. Формула для нахождения суммы n членов арифметической прогрессии в данном случае приводится к выражению 0.5·(1 + 100)·100, значение которого равно 5050. Учителя предлагают школьникам эту задачу, требуя использовать циклы и демонстрируя тем самым, что программисту не следует думать над тем, как составлять алгоритм. На их месте я бы давал оба подхода. Используем цикл **for**, поскольку количество повторений известно, а в теле цикла нужно последовательно получать значения ряда чисел 1, 2, 3, … n и суммировать их.

```
##
var s := 0;
for var i := 1 to 100 do
  s += i;
Print(s)
```

```
5050
```

Задача 4.4
Среди целых чисел из интервала [a;b] выбрать те, которые кратны трем и оканчиваются цифрой 9. Найти сумму отобранных чисел и их среднее значение.

Простейший алгоритм – перебрать все числа из интервала в цикле **for**, «просеяв» их через указанный набор условий. Накопить в переменной *s* сумму отобранные чисел, а в переменной *k* – их количество. Среднее значение *m* найти как отношение *s / k*. Значения *a* и *b* будем вводить с клавиатуры. Перед выполнением цикла полагаем *s = 0, k = 0*.

```
##
var (a, b) := ReadInteger2('Введите границы интервала:');
var (s, k) := (0, 0);
for var i := a to b do
  if (i mod 3 = 0) and (Abs(i) mod 10 = 9) then
    (s, k) := (s + i, k + 1);
Print('Сумма', s, '  среднее', s / k)
```

```
Введите границы интервала: -39 217
Сумма 645   среднее 71.6666666666667
```

Abs(i) в программе необходимо для случая, когда *a < 0*. Для отрицательных чисел операция **mod** возвращает ноль или отрицательный остаток.

Чтобы сделать алгоритм эффективнее, можно найти в начале заданного интервала число, кратное 3 и затем каждый раз прибавлять к нему по 3, обеспечивая кратность трем для всех получаемых значений. При этом проверять нужно будет только равенство девяти последней цифры в числе. Конечно, при этом мы не можем использовать цикл **for**, изменяющий счетчик на единицу, но у нас есть **while**.

```
##
var (a, b) := ReadInteger2('Введите границы интервала:');
if a mod 3 <> 0 then
  a := a > 0 ? a + 3 - a mod 3 : a - a mod 3;
var (s, k, d) := (0, 0, a);
while d <= b do
begin
  if Abs(d) mod 10 = 9 then
    (s, k) := (s + d, k + 1);
  d += 3
end;
Print('Сумма', s, '  среднее', s / k)
```

Алгоритм усложнился. Возможность допустить в нем ошибку существенно возросла. Решайте самостоятельно, тратить ли время и силы на такую оптимизацию.

Задача 4.5
Найти сумму кубов всех целых чисел от 30 до 120.

Общий член последовательности чисел $a_k = k^3$, где k = 30, 31, ... 120. Идеальная задача для цикла **for**. Тип данных – **integer**, поскольку $120^3 \ll 2.1$ миллиарда.

```
##
var s := 0;
for var k := 30 to 120 do
  s += k * k * k;
Print(s)
```

```
52518375
```

Задача 4.6
Вычислить сумму 2/3 + 3/4 + 4/5 + ... + 20/21.

Построим формулу для общего члена последовательности. В числителях – натуральные числа от 2 до 20, в знаменателях – натуральные числа, на единицу большие, чем в числителях. $a_k = k / (k + 1)$, где k = 2, 3, ... 20. Снова цикл **for** с целыми k, а вот сами значения a_k будут иметь тип **real**.

```
##
var s := 0.0; // чтобы тип был real
for var k := 2 to 20 do
  s += k / (k + 1);
Print(s)
```

```
16.8546412952373
```

Код программы в целом мало отличается от кода в предыдущей задаче. Некоторая практика позволяет программировать подобные задачи почти автоматически.

Задача 4.7
Вычислить сумму $1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\ldots+(-1)^{k+1}\frac{1}{k},\ \ k=1,2,\ldots 25$

Формула общего члена последовательности задана, количество членов n = 25. Есть одна лишь особенность – перемена знака у каждого члена. И что, будем возводить каждый раз -1 в степень k+1? Это совершенно порочная идея, реализация которой часто порождает ошибки. Имеется альтернативное и очень простое решение: заводим переменную z, хранящую знак. Первоначально z := 1, а далее в каждом проходе по циклу пишем z := -z. И используем z в качестве множителя. Не забываем, что для накопления суммы нужна переменная типа **real**. Снова берем за основу один из кодов, приведенных выше.

```
##
var (s, z) := (0.0, 1);
for var k := 1 to 25 do
begin
  s += z / k;
  z := -z
end;
Print(s)
```

```
0.712747499542829
```

В теле цикла больше одного оператора, поэтому пришлось оформить его в виде блока.

Задача 4.8

Вычислить сумму $1-x+\frac{x^3}{3}-\frac{x^5}{5}+\ldots-\frac{x^{29}}{29}$ при $x=-2.08$

Здесь формулу общего члена придется определить. Поскольку значение *x* задано, в пределах расчета можно считать *x* постоянной величиной. Каждый член содержит дробь вида x^m/m, знак перед которой меняется с минуса на плюс и обратно. Если записать x в виде $x^1/1$, закономерность видна хорошо: m = 2k – 1 при k = 1, 2, ... 15. Со знаком тоже несложно: первый минус, а дальше чередование. Определенная таким образом последовательность не порождает первого члена, равного 1. Нестрашно: исключим его из последовательности и сразу поместим в переменную для суммы.

$$1+\left(-\frac{x^1}{1}+\frac{x^3}{3}-\frac{x^5}{5}+\ldots-\frac{x^{29}}{29}\right)=1+\sum_{k=1}^{15}(-1)^k\,\frac{x^{2k-1}}{2k-1}\quad\text{при } x=-2.08$$

Все, можно писать программу? Не спешите: как собираетесь находить x^{2k-1}? Писать x ** (2 * k - 1) ? Ужасное решение. Вспомните, как мы поступили с «возведением в степень» минус единицы – всего лишь меняли знак. А тут можно брать очередное значение числителя и домножать на x^2. Сама программа строится по все той же схеме вычисления суммы.

```
##
var (x, s, z) := (-2.08, 1.0, -1);
var x2 := x * x;  // квадрат x
var p := x; // первое значение числителя
for var k := 1 to 15 do
begin
  s += z * p / (2 * k - 1);
  z := -z;
  p *= x2
end;
Print(s)
```

```
46262066.3261305
```

4.7.3. Ряды значений, вычисляемых рекуррентно

Рекуррентная формула (соотношение) определяет значение текущего члена последовательности через один или более предшествующих. При этом обязательно должны быть определены один или более начальных членов последовательности. Например, факториал числа n (произведение натуральных чисел от 1 до n) можно вычислить по следующей рекуррентной формуле: n! = n·(n – 1)!, 1! = 1. Если кроме натуральных чисел может встретиться ноль, то 0! = 1. Посмотрим, как происходит вычисление по рекуррентной формуле.

Пусть требуется вычислить 4! По формуле записываем 4! = 4 · (4 - 1)! = 4 · 3!
Теперь для 3! можно записать 3! = 3 · (3 - 1)! = 3 · 2!
Далее записываем 2! = 2 · (2 - 1)! = 2 · 1! = 2 · 1 = 2
Полученное значение используем для нахождения 3! = 3 · 2 = 6
И, наконец, 4! = 4 · 6 = 24.

В случае, когда требуется получить целый ряд факториалов натуральных чисел от 1 до n, достаточно построить цикл со счетчиком от

1 до n и находить по приведенной рекуррентной формуле члены последовательности.

Задача 4.9

Получить и вывести в строку первые m чисел Фибоначчи (m > 2). Первые два числа Фибоначчи равны единице, а каждое последующее определяется как сумма двух предыдущих, т.е. $a_k = a_{k-2} + a_{k-1}$, $k = 3, 4, \ldots m$.

Все числа целые; примем, что значение *m* небольшое и полученные значения разместятся в типе **integer**.

```
##
var m := ReadInteger('m =');
var (a, b) := (1, 1);
Print(a, b);
for var k := 3 to m do
begin
  var c := a + b;
  Print(c);
  (a, b) := (b, c)
end
```

```
m = 21
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946
```

Здесь заслуживает внимания оператор *(a, b) := (b, c)*. Кортежное присваивание, эквивалентное паре операторов присваивания *a := b; b := c*. Оно нужно для того, чтобы при очередном проходе по циклу в переменные *a* и *b* попали значения именно двух предшествующих элементов последовательности. Например, в начале *a := 1* и *b := 1.* Затем вычисляется *c = a + b* и переменная *c* получает значение 2. Теперь, чтобы получить очередной член последовательности, надо сложить 1 и 2, т.е. в формуле *a + b* переменная *a* должна получить значение, которое было в *b*, переменная *b* – только что вычисленное значение из *c*. Это типичная операция при вычислении по рекуррентным формулам.

Задача 4.10

Найти наибольшее пятизначное число Фибоначчи, Первые два числа Фибоначчи равны единице, а каждое последующее определяется как сумма двух предыдущих, т.е. $a_k = a_{k-2} + a_{k-1}$, $k = 3, 4, \ldots m$.

Проводим вычисления до тех пор, пока не получим $a_k > 99999$. Ответом будет значение числа a_{k-1}. Тип переменных – **integer**.

```
##
var (a, b) := (1, 1);
var c: integer;
repeat
  c := a + b;
  (a, b) := (b, c)
until c > 99999;
Print(a)
```

```
75025
```

Переменную *c* нужно объявить вне цикла, поскольку она используется в условии его завершения (за пределами тела цикла). Цикл завершается после выполнения конкретного условия, поэтому тут удобнее использовать **repeat**, чтобы писать **begin** ... **end**.

Задача 4.11

Найти сумму $2^2 + 2^3 + 2^4 + ... + 2^{10}$. Операцию возведения в степень не использовать.

Составляем рекуррентную формулу для k-го члена последовательности. Замечаем, что $2^3 = 2^2 \cdot 2$, $2^4 = 2^3 \cdot 2$, ... $2^k = 2^{k-1} \cdot 2$, k= 2, 3, ... 10. Отсюда получаем формулу

$$a_k = \begin{cases} 2 \cdot a_{k-1}, & \text{если } k > 2 \\ 4, & \text{если } k = 2 \end{cases} = 2 \cdot \begin{cases} a_{k-1}, & \text{если } k > 2 \\ 2, & \text{если } k = 2 \end{cases}$$

Полезно приучить себя записывать формулу суммы: так лучше виден алгоритм.

$$S = \sum_{k=2}^{10} a_k$$

```
##
var (s, a) := (0, 2);
for var k := 2 to 10 do
begin
  a *= 2;
  s += a
end;
Print(s)
```

```
2044
```

Эта программа будет работать, но при ее анализе возникает вопрос, для чего нужна переменная *k*, если она не используется в теле цикла? Ведь мы уже определились, что в таких случаях проще использовать цикл **loop**. В разделе 4.2 была приведена формула, по которой можно определить количество повторений цикла, поэтому n = 10 – 2 + 1 = 9.

```
##
var (s, a) := (0, 2);
loop 9 do
begin
  a *= 2;
  s += a
end;
Print(s)
```

Задача 4.12

Начав тренировки, лыжник в первый день пробежал 10км. Каждый следующий день он увеличивал пробег на 10% от пробега предыдущего дня. Определить:

а) пробег лыжника за второй, третий, ... десятый день тренировок;

б) какой суммарный путь пробежал лыжник за первые 7 дней тренировок.

Ищем рекуррентную формулу для k-го дня. Обозначаем пробег *k*-го дня через a_k. В соответствии с условием задачи, a_1 = 10.0км, тогда $a_2 = a_1 + 10 / 100 \cdot a_1 = 1.1a_1$. $a_3 = 1.1a_2$, ... $a_k = 1.1a_{k-1}$. Решение очень похоже на предыдущее.

```
##
var (s, a) := (10.0, 10.0);
Writeln('День   Пробег, км');
//       *XX******XX.XXX - для расчета позиций вывода
for var k := 2 to 10 do
begin
  a *= 1.1;
  Writeln(k:3, a:12:3);
  if k <= 7 then
    s += a
end;
Writeln;
Write('Суммарный пробег за первых 7 дней, км: ', s:0:3)
```

```
День    Пробег, км
  2       11.000
  3       12.100
  4       13.310
  5       14.641
  6       16.105
  7       17.716
  8       19.487
  9       21.436
 10       23.579
Суммарный пробег за первых 7 дней, км: 94.872
```

В переменную *s* не забываем поместить пробег первого дня, равный 10 км – это частая ошибка, допускаемая в задачах такого рода. Ведь в цикле обрабатываются дни, начиная со второго.

4.7.4. Анализ цифр в произвольном целом числе

Под произвольным целым пока условимся понимать такое число, для размещения которого достаточно переменной типа **integer**. Его цифры получают в обратном порядке (начиная с разряда единиц) посредством цикла с неизвестным количеством повторений. Если число *n* может быть отрицательным, его следует взять по абсолютной величине при помощи функции *Abs(n)*.

Задача 4.13
Найти сумму и произведение всех цифр в целом числе, введенном с клавиатуры.

Цифры числа получаем по алгоритму, приведенному в приложении П5. Сумму цифр *S* и произведение *P* получаем по стандартным алгоритмам П4. Поскольку не сказано, что число натуральное, следует допустить, что оно может быть и отрицательным.

```
##
var n := Abs(ReadInteger('Введите число:'));
var (S, P) := (0, 1);
while n > 0 do
begin
  var d := n mod 10; // очередная цифра числа
  S += d;
  P *= d;
  n := n div 10
end;
Write('Сумма цифр ', S, ', произведение ', P)
```

```
Введите число: 1583751
Сумма цифр 30, произведение 4200
```

Задача 4.14

Дано натуральное число. Верно ли, что сумма его четных цифр больше, чем сумма нечетных?.

Цифры числа получаем по алгоритму, приведенному в приложении П5. Нужно накапливать две суммы – нечетных цифр *S1* и четных *S2*. Четность числа *k* определяется истинностью выражения *k* ***mod*** *2 = 0* либо истинностью выражения *k.IsEven*. Если оперировать нечетностью числа, можно использовать *k* ***mod*** *2 <> 0,* либо *k.IsOdd*, либо *Odd(k).* Не нужно приучать себя к проверке нечетности числа по условию k **mod** 2 = 1. Для отрицательных нечетных k получается значение -1, а не 1. И пусть даже в данном случае цифры всегда положительны, не нужно привыкать пользоваться некорректными алгоритмами.

```
begin
  var n := ReadInteger('Введите число:');
  var (S1, S2) := (0, 0);
  while n > 0 do
  begin
    var d := n mod 10; // очередная цифра числа
    if d.IsOdd then
      S1 += d
    else
      S2 += d;
    n := n div 10
  end;
  if S2 > S1 then
    Print('Верно')
  else
    Print('Неверно')
end.
```

```
Введите число: 246831
Верно
```

Задача 4.15

Дано натуральное число. Верно ли, что в нем заданная цифра встречается более одного раза?

Цифры числа *n* получаем по алгоритму, приведенному в приложении П5. Заводим счетчик *k*, в который будем добавлять по единице всякий раз, когда встретим искомую цифру *a*. При значении *k = 2* досрочно выходим из цикла.

```
begin
  var n := ReadInteger('Введите число:');
  var a := ReadInteger('Какую цифру ищем?');
  var k := 0;
  while n > 0 do
  begin
    if n mod 10 = a then
      k += 1;
    if k = 2 then
      break;
    n := n div 10
  end;
  if k > 1 then
    Print('Верно')
  else
    Print('Неверно')
end.
```

```
Введите число: 252333254
Какую цифру ищем? 2
Верно
```

4.7.5. Табуляция функции одной переменной

Задача получения таблицы значений некоторой функции для значений аргумента на заданном интервале достаточно распространена. Рассмотрим ее реализацию с помощью операторов цикла.

Пусть требуется составить таблицу значений функции y = F(x) для x, меняющегося от a до b с шагом h. Часто для указания подобных условий используют запись вида a(h)b, например -10(0.25)12. Здесь x последовательно получает значения из ряда a, a+h, a+2h, … b и для каждого x нужно найти соответствующее значение функции y = F(x). Мы уже работали с подобными последовательностями чисел в подразделе 4.7.2, но там мы находили различные суммы и произведения, а здесь будем вычислять значения *y*.

Использование цикла for потребует некоторых предварительных расчетов. Как было показано, очередное значение x зависит от номера свой позиции в последовательности. Цикл **for** обеспечивает изменение своего целочисленного параметра на 1 или -1, что и задает требуемый номер. Остается найти зависимость, связывающую номер позиции с величиной x.

Определим, сколько раз должен выполняться цикл:

$$n = \left\lfloor \frac{b-a}{h} \right\rfloor + 1; \quad n := Trunc((b-a)/h) + 1;$$

Заголовок цикла записываем в виде **for var** i := 0 **to** n-1 **do**. Значения *x* в этом случае получаются очень просто: **var** x := a + i * h.

Шаблон для кода задачи табуляции на основе цикла **for** можно записать следующим образом:

```
var (a, b, h) := ReadReal3;
for var i := 0 to Trunc((b - a) / h) do
begin
  var x := a + i * h;
  // тут находим y(x)
end;
```

Использование цикла while обычно предполагает реализацию табуляции «в лоб». Тело цикла выполняется для начального значения, потом на каждом проходе значение наращивается на шаг. Контролируется достижение верхней границы. Предлагается следующий шаблон кода:

```
var (a, b, h) := ReadReal3;
var x := a;
while x <= b do
begin
  // тут находим y(x)
  x += h
end;
```

У такой реализации имеется недостаток, который проявляется при работе с некоторыми значениями типа **real**. Он связан с неточностью представления вещественных чисел в компьютере. Многократное сложение увеличивает начальную погрешность и в результате значение *x* может не прийти в точку *b*. В связи с этим алгоритм улучшают, используя условие *x <= b + h/2* (или *h/3*, *h/4* и т.п.).

```
var (a, b, h) := ReadReal3;
var x := a;
while x <= b + h / 2 do
begin
  // тут находим y(x)
  x += h
end;
```

Использование цикла repeat оригинальностью не отличается, поскольку эта разновидность цикла взаимозаменяема с **while**.

```
var (a, b, h) := ReadReal3;
var x := a;
repeat
  // тут находим y(x)
  x += h
until x > b + h / 2;
```

Задача 4.16

Составить таблицу значений функции $y = 3x^2 + 2x - 4$ для x, изменяющегося в диапазоне от 0 до 10 с шагом 0.5.

Реализация на основе цикла **for**:

```
##
var (a, b, h) := (0.0, 10.0, 0.5);  // все типа real
for var i := 0 to Trunc((b - a) / h) do
begin
  var x := a + i * h;
  var y := 3 * x * x + 2 * x - 4;
  Writeln(x:4:1, y:10:2)
end
```

```
 0.0     -4.00
 0.5     -2.25
 ...
10.0    316.00
```

Реализация на основе цикла **repeat**:

```
##
var (a, b, h) := (0.0, 10.0, 0.5);  // все типа real
var x := a;
repeat
  var y := 3 * x * x + 2 * x - 4;
  Writeln(x:4:1, y:10:2);
  x += h
until x > b + h / 2
```

Задача 4.17

Составить таблицу значений функции для x, изменяющегося в диапазоне от -1 до 5.2 с шагом 1.2.

$$y = \begin{cases} 2, & \text{при } x \le 1 \\ \sqrt[3]{x}, & \text{при } 1 < x \le 3 \\ (x-5)^2 + 1, & \text{при } x > 3 \end{cases}$$

```
##
var (a, b, h) := (-1.0, 5.2, 1.2);  // все типа real
for var i := 0 to Trunc((b - a) / h) do
begin
  var x := a + i * h;
  var y: real;
  if x <= 1 then
    y := 2
  else if x <= 4 then
    y := x ** (1 / 3)
  else
    y := Sqr(x - 5) + 1;
  Writeln(x:4:1, y:15:7)
end
```

```
-1.0      2.0000000
 0.2      2.0000000
 1.4      1.1186889
 2.6      1.3750689
 3.8      1.5604908
 5.0      1.0000000
```

В подобных случаях условное выражение существенно сокращает код программы. Но при этом делает его нескольно менее наглядным.

```
##
var (a, b, h) := (-1.0, 5.2, 1.2);  // все типа real
for var i := 0 to Trunc((b - a) / h) do
begin
  var x := a + i * h;
  var y := x <= 1 ? 2 : x <= 4 ? x ** (1 / 3) : Sqr(x - 5) + 1;
  Writeln(x:4:1, y:15:7)
end
```

Задача 4.18

Для функции $y = x^4 - 4.1x^3 + x^2 - 5.1x + 4.1$ найти минимальное значение на интервале изменения x от -4 до 5. Требуется получить точность по аргументу не ниже четырех знаков после запятой.

Такую задачу можно решить при помощи табуляции. Самое простое решение – пройти весь интервал с шагом 0.0001 и найти точку минимума функции. На современных компьютерах выполнить цикл (5 + 4) / 0.0001 + 1 = 9001 раз – совершенно не проблема. Можно порассуждать об эффективности такого решения, подискутировать о различных методах, а можно молча потратить пару минут и получить решение, оставив болтовню (и проблемы) другим.

Функция представляет собой полином четвертой степени, поэтому для вычисления ее значений можно использовать схему Горнера (Приложение П6).

$y = x^4 - 4.1x^3 + x^2 - 5.1x + 4.1 = 4.1 + x(-5.1 + x(1 + x(-4.1 + x)))$

Алгоритм поиска минимума – стандартный. Полагаем минимум *ym* равным максимально возможному для типа **real** значению, а затем, если очередное полученное значение функции *y* оказывается меньше *ym*, полагаем *ym* равным *y*. В программе нужно будет также хранить значение аргумента *x*, при котором минимум был достигнут. Для этого в программе введем переменную *xm*.

```
##
var (a, b, h) := (-4.0, 5.0, 0.0001);
var (xm, ym) := (a - 1, real.MaxValue);
for var i := 0 to Trunc((b - a) / h) do
begin
  var x := a + i * h;
  var y := 4.1 + x * (-5.1 + x * (1 + x * (-4.1 + x)));
  if y < ym then
    (xm, ym) := (x, y)
end;
Print('Минимум', ym, 'достигается при x =', xm)
```

```
Минимум -31.9438193152287 достигается при x = 3.0482
```

Для интереса я замерил время выполнения программы (без выдачи результата) – оно составило 1.7 миллисекунды. Стоит ли рассуждать об эффективности такого алгоритма и думать как сэкономить тысячную долю секунды? Как видите, и простое решение может быть вполне удовлетворительным.

4.7.6. Табуляция функции двух переменных

Если функция имеет два аргумента, при табуляции приходится перебирать все их сочетания. На практике фиксируется значение одного аргумента, и функция вычисляется для всего набора значений другого аргумента. Затем зафиксированный ранее аргумент меняется на величину шага и вычисления функции повторяются для всего набора значений другого аргумента. Процесс завершается после полного перебора комбинаций значений обоих аргументов.

Реализация такой схемы табуляции производится при помощи двух циклов, один из которых является вложенным внутрь другого. Для каждого из значений параметра внешнего цикла во внутреннем (вложенном) цикле перебираются все значения параметра внутреннего цикла. На базе параметров обоих циклов строятся значения аргументов.

Примером табуляции функции двух переменных является таблица умножения, которую должен знать каждый. Посмотрите, как просто ее получить.

```
##
for var i := 1 to 9 do
begin
  Write(i);
  for var j := 1 to 9 do
    Write(i * j:4);
  Writeln
end
```

```
1   1   2   3   4   5   6   7   8   9
2   2   4   6   8  10  12  14  16  18
3   3   6   9  12  15  18  21  24  27
4   4   8  12  16  20  24  28  32  36
5   5  10  15  20  25  30  35  40  45
6   6  12  18  24  30  36  42  48  54
7   7  14  21  28  35  42  49  56  63
8   8  16  24  32  40  48  56  64  72
9   9  18  27  36  45  54  63  72  81
```

Задача 4.19

Составить таблицу значений функции F(x,y) для x, меняющегося от -2π до 3π с шагом π/6 и y, меняющегося от -1.2 до 5.9 с шагом 0.4.

$$F = \frac{1.4e^{-1.6y} \cdot \cos^2\left(\frac{\pi x}{2}\right)}{1+0.17\sqrt[3]{x-1.185}}$$

Для каждого значения *y* будем задавать перебор всех значений *x*, тогда внешний цикл будет менять значение *y*, а внутренний – значение *x*.

Функция содержит «подарок» для неопытных программистов – извлечение кубического корня. До сих пор мы находили такие корни путем возведения подкоренного выражения в степень 1/3. Проблем не возникало, потому что во всех случаях подкоренное выражение было положительным. А в данном случае возможно появление под корнем отрицательных значений. В математике кубический корень (а также иной с нечетной степенью) определен и для отрицательного аргумента, а вот операция возведения отрицательного значения в нецелую степень в PascalABC.NET дает NaN (в предыдущих версиях Паскаля программа просто «слетит»). Поэтому будем использовать абсолютную величину аргумента, а знак сформируем посредством функции Sign(x).

```
##
var (ax, bx, hx) := (-2 * Pi, 3 * Pi, Pi / 6);
var (ay, by, hy) := (-1.2, 5.9, 0.4);
for var j := 0 to Trunc((by - ay) / hy) do
begin
  var y := ay + hy * j;
  for var i := 0 to Trunc((bx - ax) / hx) do
  begin
    var x := ax + hx * i;
    var p := 1.4 * Exp(-1.6 * y) * Sqr(Cos(Pi * x / 2));
    var p1 := x - 1.185;
    p1 := 1 + 0.17 * Sign(p1) * Abs(p1) ** (1 / 3);
    p := p / p1;
    Println(y, x, p)
  end
end
```

Числитель *p* и знаменатель *p1* для удобства записи вычисляются раздельно.

4.7.7. Нахождение НОД и НОК

Наиболее популярный в школе способ нахождения НОД – наибольшего общего делителя двух чисел – состоит в использовании алгоритма Евклида. Классический алгоритм Евклида реализован с помощью вычитания, но обычно применяют его в модификации, использующей деление.

1. Делим большее из чисел на меньшее, получая остаток.
2. Если остаток нулевой, меньшее число принимаем за НОД и завершаем алгоритм.
3. Большее число заменяем на остаток от деления.
4. Переходим к 1.

С использованием кортежного присваивания алгоритм Евклида реализуется в PascalABC.NET фактически в две строки:

```
##
var (a, b) := ReadInteger2;
while b <> 0 do
  (a, b) := (b, a mod b);
Print(a)
```

Произведение НОД и НОК (наименьшего общего кратного) чисел *a* и *b* равно их произведению, поэтому зная НОД, можно найти НОК:

$$НОК = \frac{a \cdot b}{НОД}$$

Приведенные выше алгоритм, программа и формула предполагают, что числа a и b – натуральные.

Задача 4.20
Найти НОК трех чисел a, b и c, введенных с клавиатуры.

Найдем НОК(a, b) и заменим им значение a. Затем найдем НОК(a, c).

```
##
var (a, b, c) := ReadInteger3;
var (pa, pb) := (a, b); // a и b не сохраняются
while b <> 0 do
  (a, b) := (b, a mod b);
a := pa div a * pb; // НОК(a, b)
(pa, pb) := (a, c);
while c <> 0 do
  (a, c) := (c, a mod c);
a := pa div a * pb;
Print(a)
```

```
12 4 20
60
```

Задача 4.21

Даны две простые дроби a/b и c/d, где a, b, c, d – натуральные числа. Представить их сумму несократимой простой дробью, либо целым числом.

Сложение простых дробей можно выполнить по следующей формуле

$$\frac{a}{b}+\frac{c}{d}=\frac{ad+bc}{bd}=\frac{p}{q}$$

Затем найдем k = НОД(p, q) и разделим на него p и q, получая несократимую дробь. Если q = 1, то дробь превращается в целое число.

```
##
var (a, b, c, d) := ReadInteger4;
Write(a, '/', b, ' + ', c, '/', d, ' = ');
var p := a * d + b * c;
var q := b * d;
(a, b) := (p, q);
while b <> 0 do
  (a, b) := (b, a mod b);
p := p div a;
q := q div a;
if q <> 1 then
  Write(p, '/', q)
else
  Write(p)
```

```
3 8 11 24
3/8 + 11/24 = 5/6
```

4.7.8. Простые числа

Простым в математике называют натуральное число, которое имеет только два делителя – единицу и собственное значение.

Тривиальный алгоритм проверки на простоту – поочередно делить число на 2, 3, 4 и т.д., пока не достигнем самого числа или не найдем делителя. Но это плохое решение. В самом деле, если число нечетное, зачем проверять четные делители – а это ведь половина чисел! Если число не делится на 3, не нужно проверять делители кратные 3 и т.д. Далее, незачем проверять делители, которые превышают половину числа, поскольку на них деление нацело невозможно. И более того, оказывается, что достаточно проверять делители лишь до значения, равного квадратному корню из числа. Подобный алгоритм с перебором делителей предлагается для проверки на числа на простоту.

Задача 4.22
Является ли заданное натуральное число простым?

```
##
var n := ReadInteger;
var (j, IsPrime) := (2, True);
while j * j <=n do
begin
  if n mod j = 0 then
  begin
    IsPrime := False;
    break
  end;
  j += 1
end;
Print(IsPrime and (n > 1) ? 'Простое' : 'Не простое');
```

```
7351
Простое
```

Отметим, что предложенный код можно улучшить, выделив проверку для простого числа 2, а затем начать с 3 и двигаться с шагом 2.

Задача 4.23
Получить сто первых простых чисел.

```
begin
  var (n, k) := (1, 0);
  repeat
    var r: boolean; // False если число не простое
    if n <= 1 then
      r := False
    else if n = 2 then
      r := True
    else if n mod 2 = 0 then
      r := False
    else
    begin
      r := True;
      var j := 3;
      while (j * j <= n) and r do
        if n mod j = 0 then
          r := False
        else
          j += 2
    end;
    if r then
    begin
      k += 1;
      Print(n)
    end;
    n += 1
  until k > 100 // требуются 100 простых
end
```

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83
89 97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173
179 181 191 193 197 199 211 223 227 229 233 239 241 251 257 263
269 271 277 281 283 293 307 311 313 317 331 337 347 349 353 359
367 373 379 383 389 397 401 409 419 421 431 433 439 443 449 457
461 463 467 479 487 491 499 503 509 521 523 541 547
```

Встроить реализацию алгоритма проверки на простоту в цикл помогла логическая переменная Обратите внимание на логическое выражение *(j * j <= n)* ***and*** *r*. Поскольку *r* имеет логический тип, писать что-то наподобие *(r = True)* излишне. Сама же задача тривиальна, если алгоритм проверки на простоту уже реализован. Перебираем в цикле числа натурального ряда и если очередное число простое –

выводим его. Цикл завершается, как только будут найдены сто чисел – за этим следит счетчик *k*.

Существует древний, но при этом красивый алгоритм получения простых чисел, называемый «Решето Эратосфена». К сожалению, пока мы не можем его реализовать, поскольку там используются массивы, а они не входят в первую часть книги.

4.7.9. Задача о сдаче («жадный» алгоритм)

Пусть в некотором государстве находятся в обращении монеты достоинством $a_1 < a_2 < \ldots < a_n$. Требуется выдать при помощи этих монет сумму P так, чтобы количество монет было минимальным. При этом гарантируется, что существует минимум одно решение задачи. Для поиска решения в общем случае используются методы ***динамического программирования***, которыми славятся некоторые олимпиадные задачи. В частных случаях, когда ряд значений a_1, a_2, … a_n образует так называемую каноническую систему (она характерна для реальных наборов достоинств монет различных стран) решение может быть найдено более простыми методами, в том числе, с помощью ***«жадного» алгоритма***.

В основе «жадного» алгоритма лежит принцип локальной оптимизации, полагающий что на каждом шаге принимаемое решение является оптимальным и ведущим к получению общего оптимального решения. Применительно к задаче о сдаче такая стратегия состоит в последовательной выдаче частичных сумм наибольшим возможным количеством монет наибольшего достоинства.
1. Положить k = n
2. Найти m = P div a_k
3. Если m > 0, выдать m монет достоинством a_k, положить P = P mod a_k. Если P = 0, завершить алгоритм.
4. Уменьшить k на 1 и перейти к 1.

Рассмотрим пример. Пусть P = 24, а достоинства монет составляют ряд 1, 5, 10, 50.
- 24 div 50 = 0, монеты достоинством 50 не используются

- 24 div 10 = 2, выдаем 2 монеты достоинством 10, P полагаем равным 24 mod 10 = 4
- 4 div 5 = 0, монеты достоинством 5 не используются
- 4 div 1 = 4, выдаем 4 монеты достоинством 1, P полагаем равным 4 mod 1 = 0. Алгоритм завершен.

Сумма P = 24 выдается двумя монетами достоинства 20 и четырьмя монетами достоинства 1, всего шестью монетами. Это оптимальное решение, все другие дадут большее количество монет.

Теперь допустим, что достоинства монет представлены неканоническим рядом 1, 5, 7. При помощи «жадного» алгоритма получаем три монеты достоинства 7 и три достоинства 1 – всего шесть монет. На самом деле лучшее решение – две монеты достоинства 7 и две достоинства 5 – всего четыре монеты. Этот пример показывает, что «жадный» алгоритм применим не всегда.

Ряд достоинств монет может считаться каноническим, если соседние значения в нем отличаются не менее, чем вдвое.

Реализация жадного алгоритма достаточно проста. Алгоритм циклический, но проблема в том, что при очередном проходе по циклу нужно переходить к очередному достоинству монеты, а это можно сделать лишь с использованием последовательности или массива – информационных структур, которые будут рассматриваться в следующей книге. Поэтому пока что используем, как часто говорят программисты, «костыль» – значение достоинств монет будем вводить с клавиатуры в каждом проходе по циклу.

Задача 4.24

Выдать указанную сумму минимальным количеством монет достоинства 50, 10, 5 и 1. Количество монет каждого достоинства не ограничивается.

```
begin
  var P := ReadInteger('Укажите сумму:');
  var n := 0;
  repeat
    var a := ReadInteger('Укажите достоинство монеты:');
    var m := P div a;
    if m > 0 then
    begin
      Println('Выдаем', m, 'монет достоинства', a);
      P := P mod a;
      n += m;
      if P = 0 then
        break // прервать цикл
    end;
  until False; // бесконечный цикл
  Print('Сумма P выдана, количество монет равно', n)
end.
```

```
Укажите сумму: 24
Укажите достоинство монеты: 50
Укажите достоинство монеты: 10
Выдаем 2 монет достоинства 10
Укажите достоинство монеты: 5
Укажите достоинство монеты: 1
Выдаем 4 монет достоинства 1
Сумма P выдана, количество монет равно 6
```

Проверим, как работает алгоритм для неканонического ряда.

```
Укажите сумму: 24
Укажите достоинство монеты: 7
Выдаем 3 монет достоинства 7
Укажите достоинство монеты: 5
Укажите достоинство монеты: 1
Выдаем 3 монет достоинства 1
Сумма P выдана, количество монет равно 6
```

4.8. Использование случайных чисел

Среди стандартных функций (подраздел 1.5.4) упоминалась группа функций с именем *Random*. Они возвращают одно псевдослучайное число (*Random*), или кортеж из двух (*Random2*), либо из трех (*Random3*) таких чисел. Псевдослучайные числа получаются при помощи так называемого генератора псевдослучайных чисел (ГПСЧ).

Этот генератор при каждом обращении выдает очередное число. Существует стартовое значение для инициализации ГПСЧ и его автоматически вырабатывает среда .NET. Процедура *Randomize* инициализирует ГПСЧ некоторым значением. Для целей отладки можно использовать вызов *Randomize(n)*, где *n* – некоторое целое число, инициализирующее ГПСЧ. Это обеспечит порождение одной и той же последовательности случайных чисел, что и требуется для отладки программы.

Функция Random может возвращать как целое число, так и вещественное. Целые числа всегда возвращаются на отрезке *[m; n]*, вещественные – на промежутке *[a; b)*, так что *Random(a, b)* никогда не вернет вещественное псевдослучайное значение, точно равное *b*.

Псевдослучайные числа с успехом заменяют ввод данных и часто используются для демонстрации работы программ. Кроме того, такие числа используются в алгоритмах моделирования и некоторых методах вычислений, например, в методе Монте-Карло.

Задача 4.25
Даны 10 000 троек вещественных чисел, принадлежащих промежутку [10; 99). Найти и вывести с точностью два знака после запятой процент троек, из которых можно построить треугольники.

Безусловно, эта задача предполагает использование случайных чисел, ведь никто не станет придумывать и вводить 30 000 значений. Проверка возможности построения треугольника из трех прямых отрезков заданной длины была рассмотрена в задаче 3.16 (подраздел 3.5.6). Строим цикл на 10 000 повторений. Значение параметра цикла не используется, поэтому выберем цикл **loop**. Тройки случайных чисел будем получать посредством вызова Random3(10.0, 99.0). Вещественные значения границ обеспечат получение значений типа **real**.

```
##
var k := 0; // количество составленных треугольников
var n := 10000; // общее количество треугольников
loop n do
begin
  var (a, b, c) := Random3(10.0, 99.0);
  if (a + b > c) and (a + c > b) and (b + c > a) then
    k += 1
end;
Write(k * 100 / n:0:2, ' %')
```

```
64.86 %
```

Поскольку значения случайные, результаты могут несколько разниться при каждом запуске, но при этом все равно будут принадлежать отрезку [64; 66]. Вы можете проверить результаты и для других n, например для ста тысяч или миллиона – программа работает очень быстро. Напротив, для малых n результаты будут сильно разниться. Это особенность работы со случайными числами – чем меньше данных, тем непредсказуемее результат.

Примите поздравления: вы только что решили задачу по теории вероятности методом Монте-Карло! Этот метод предполагает многократное моделирование процесса при помощи случайных чисел и получение результата как отношения числа успешного моделирования к общему числу испытаний.

Задача 4.26

Душевая кабинка на виде сверху имеет форму квадрата со стороной в один метр. Начертим на полу кабинки круг максимального диаметра. Считая, что вода из лейки душа равномерно попадает во все точки круга (и не выходит за его пределы), определите бесполезную часть площади кабинки.

Задача легко решается аналитически. Пусть сторона кабинки имеет длину *a*, тогда ее площадь равна a^2, т.е. 1. Круг максимального диаметра (d = 1) имеет площадь π/4, разница площадей составит 1 – π/4, а потери площади $(1 - \pi/4)/1 \approx 0.2146$. Это решение приведено с тем, чтобы потом оценить точность решения задачи методом Монте-Карло.

```
##
var k := 0; // количество попаданий капель воды в круг
var n := 1000000; // общее количество капель воды
loop n do
begin
  var (x, y) := Random2; // промежуток по умолчанию [0.0; 1.0)
  if x * x + y * y <= 1 then // условие попадания в круг
    k += 1
end;
Write((n - k) / n:0:4)
```

```
0.2145
```

Неплохо, не так ли?

Задача 4.27
Вычислить число π методом Монте-Карло.

Обратимся к предыдущей задаче. Площадь квадрата со стороной единичной длины численно равна 1. Площадь круга, вписанного в этот квадрат, численно равна π/4. Отношение площади круга к площади квадрата *p* также равно π/4, поэтому для получения значения π достаточно вычислить величину *4p*.

```
##
var k := 0; // количество попаданий точек в круг
var n := ReadInteger('Количество точек:');
loop n do
begin
  var (x, y) := Random2; // промежуток по умолчанию [0.0; 1.0)
  if x * x + y * y <= 1 then // условие попадания в круг
    k += 1
end;
Write(4 * k / n:0:4)
```

```
Количество точек: 1000000
3.1414
```

Мы нашли таинственное число π за секунду и не выходя за рамки четырех действий арифметики. Надеюсь, метод Монте-Карло вам понравился.

4.9. Перевод между системами счисления

К сожалению, пока мы будем вынуждены ограничиться системами счисления с основаниями от 2 до 10, числа в которых записываются с использованием привычных цифр. Системы счисления с основаниями, превышающими 10, будут рассмотрены в следующей книги после ознакомления с символами и строками.

4.9.1. Перевод в десятичную систему

Число в системе счисления по основанию N может быть записано в расширенном виде $M_N = a_0N^k + a_1N^{k-1} + \ldots + a_{k-1}N + a_k$

Такая запись дает возможность простого перевода числа из системы по основанию N в десятичную: достаточно представить N и коэффициенты *a* в десятичной системе, а затем вычислить результат.

Для примера, переведем в десятичную систему счисления число 6537303_8.

$3 + 0\cdot8^1 + 3\cdot8^2 + 7\cdot8^3 + 3\cdot8^4 + 5\cdot8^5 + 6\cdot8^6 = 1752771_{10}$.

Подобное вычисление делается в цикле. На каждом шаге нужна степень основания, на единицу большая, чем в предыдущем проходе по телу цикла, следовательно нужно хранить некий множитель k и домножать его на основание N, получая очередную степень N.

Можно ли, пользуясь этим алгоритмом, перевести число из десятичной системы счисления в другую – систему счисления по основанию N? Безусловно. Только для этого вы должны уметь вести расчеты в такой системе счисления. Поэтому перевод в систему счисления по основанию N выполняется по другому алгоритму, который мы рассмотрим далее.

Задача 4.28

Написать программу перевода натурального числа из системы счисления по основанию N < 10 в десятичную.

```
##
var (m, n) := ReadInteger2('Укажите число и систему счисления:');
var (k, s) := (1, 0);
while m > 0 do
begin
  s += k * (m mod 10);
  k *= n;
  m := m div 10
end;
Print(s)
```

```
Укажите число и систему счисления: 6537303 8
1752771
```

В программе использован алгоритм получения цифр числа, приведенный в приложении П5.

4.9.2. Перевод в систему по основанию N

Алгоритм перевода натурального числа из десятичной системы счисления в систему по основанию N известен. Нужно делить заданное число на основание системы счисления и записывать полученные остатки в обратном порядке (справа налево). При этом за очередное делимое берется результат деления. Процесс повторятся до тех пор, пока в результате деления не получится ноль.

Рассмотрим пример. Переведем 5391 в систему счисления по основанию 8.
5931 div 8 = 741; 5931 mod 8 = 3. Пишем 3, это последняя цифра;
741 div 8 = 92; 741 mod 8 = 5. Приписываем слева 5, получая 53;
92 div 8 = 11, 92 mod 8 = 4. Приписываем слева 4, получая 453;
11 div 8 = 1, 11 mod 8 = 3. Приписываем слева 3, получая 3453;
1 div 8 = 0, 1 mod 8 = 1. Приписываем слева 1, получая 13453.
Результат деления нулевой, алгоритм завершен. $5931 = 13453_8$

К сожалению, мы пока не сможем воспользоваться этим алгоритмом, поскольку не умеем хранить отдельные цифры (но научимся в следующей части книги). Вот если бы получать цифры числа п обычном порядке, слева направо, мы могли бы их выводить по одной. Давайте попробуем построить такой алгоритм.

Снова вернемся у расширенной записи числа.
$M_N = a_0N^k + a_1N^{k-1} + \ldots + a_{k-1}N + a_k$

Нам надо получить последовательно цифры a_0, a_1, … a_{k-1}, a_k. Разделим число на N^k.

$$\frac{a_0N^k + a_1N^{k-1} + \ldots + a_{k-1}N + a_k}{N^k} = a_0 + \frac{a_1N^{k-1} + \ldots + a_{k-1}N + a_k}{N^k}$$

Получены первая цифра результата a_0 и остаток $a_1N^{k-1} + \ldots + a_{k-1}N + a_k$. Далее остаток разделим на N^{k-1}, получая a_1 и следующий остаток. Процесс продолжаем до тех пор, пока не дойдем до $N^0 = 1$, что даст последнюю цифру a_k. Проблема тут одна: чтобы начать алгоритм, нужно знать, чему равно значение k. Найти k можно разными путями. Примем k = 1 и будем последовательно умножать его на N до тех пор, пока значение k не превысит заданного числа M. Как только превысило, разделим k на N и получим первый делитель N_k для начала алгоритма. А затем будем на каждом шаге делить k на N, получая делители.

Наш алгоритм не годится для M = 1, но единица по любому основанию N > 1 всегда остается единицей, так что в этом случае ничего делать не нужно.

В подразделе 1.5.1 отмечалось, что в программе имена можно записывать с помощью символов кириллицы, но примеров таких программ пока не было. Давайте восполним этот пробел.

Задача 4.29

Написать программу перевода натурального числа из десятичной системы счисления в систему по основанию N < 10.

```
begin
  var Число := ReadInteger('Какое число переводим?');
  var Основание := ReadInteger('Какое основание требуется?');
  if Число < 2 then
  begin
    Write(Число, ' = ', Число, '(', Основание, ')');
    exit
  end;
  var Предел := integer.MaxValue div Основание;
  if Число > Предел then
  begin
    Print('Число слишком велико');
    exit
  end;
  Write(Число, ' = ');
  var (Степень, Делитель) := (0, 1);
  while Делитель < Число do
  begin
    Делитель *= Основание;
    Степень += 1
  end;
  Делитель := Делитель div Основание;
  Степень -= 1;
  for var i := Степень downto 0 do
  begin
    Write(Число div Делитель);
   (Число, Делитель) := (Число mod Делитель, Делитель div Основание)
  end;
  Write('(', Основание, ')')
end.
```

```
Какое число переводим? 1093728
Какое основание требуется? 7
1093728 = 12203466(7)
```

Если вы внимательно изучили алгоритм, то должны оценить преимущества использования имен на русском языке. Соглашусь, что набирать неудобно, поскольку имена на кириллице, остальное – на латинице, но зато удобно читать и отлаживать. И при необходимости вносить изменения, особенно по прошествии времени

По сравнению с составленным алгоритмом в программе добавлен контроль на арифметическое переполнение для значений типа **integer**, которое приведет к зацикливанию программы.

4.10.Нахождение делителей числа

Встречаются задачи на нахождение как всех делителей числа, так и только простых. При этом нужно обращать внимание на условие: единица и само число могут учитываться в качестве делителей, но могут и не учитываться.

4.10.1. Нахождение всех делителей числа

Конечно, можно использовать тривиальный алгоритм с последовательным перебором делителей из всех чисел натурального ряда, не превышающих половину значения заданного числа, но это неэффективно по времени работы. И опять эффективная реализация невозможна без знаний той части PascalABC.NET, которая будет рассмотрена во второй книге. Поэтому сначала рассмотрим неэффективную реализацию, зато дающую все делители по порядку.

Задача 4.30
Найти все без исключения делители заданного числа и вывести их в порядке возрастания.

```
##
var n := ReadInteger('n=');
Writeln('Делители: ');
for var i := 1 to n div 2 do
  if n mod i = 0 then
    Print(i);
Write(n)
```

```
n= 37532
Делители:
1 2 4 11 22 44 853 1706 3412 9383 18766 37532
```

Улучшенный алгоритм исходит из того, что если число *n* имеет делитель *m*, то оно также имеет делитель, равный *n* ***div*** *m*. Это позволяет находить сразу два делителя и вести перебор не до *n* ***div*** *2*, а до *Sqrt(n)*. Проблема в том, что при этом делители оказываются неупорядоченными. Но существуют задачи, где этого и не требуется.

Задача 4.31
Найти количество нечетных делителей заданного числа.

```
##
var n := ReadInteger('n=');
var k := 0;
for var i := 1 to Round(Sqrt(n)) do
  if n mod i = 0 then
  begin
    k += i.IsOdd ? 1 : 0;
    var t := n div i;
    k += t.IsOdd and (t <> i) ? 1 : 0;
  end;
Print(k)
```

```
n= 37532
4
```

При составлении программы учитываем, что число может быть полным квадратом. В этом случае делители *n* и *n* ***div*** *m* совпадают. Также, они совпадают для n = 1.

4.10.2. Нахождение простых делителей числа

Эту проблему часто связывают с разложением числа на простые множители. Например, 24 = 2·2·2·3 – это разложение числа 24 на простые множители. Если одинаковые множители записать с использованием операции возведения в степень ($24 = 2^3 \cdot 3$), получим так называемую факторизацию числа, и это еще одна из задач, связанных с нахождением простых делителей. Большинство эффективных алгоритмов используют запоминание промежуточных значений в массивах или иных структурах, поэтому здесь будет рассмотрено простейшее, пусть и неэффективное решение.

Задача 4.32
Найти все простые делители заданного числа.

```
##
var n := ReadInteger('n =');
var i := 2;
while i <= Round(Sqrt(n)) do
  if n mod i = 0 then
  begin
    Print(i);
    n := n div i
  end
  else
    i += 1;
if n > 1 then
  Print(n)
```

```
n = 24
2 2 2 3
```

Эту программу можно ускорить вдвое, если отдельно рассмотреть случай для делителя, равного 2, а затем положить *i := 3* и двигаться с шагом 2, указав *i += 2*. Попробуйте это сделать самостоятельно.

4.11. Решение задачи с сайта //acmp.ru

Красноярский сайт //acmp.ru достаточно хорошо известен. На нем проводятся олимпиады, а в промежутке каждый желающий может потренироваться в решении задач. Написанная на выбранном языке программа проходит автопроверку с использованием значительного количества тестов. Обстановка практически олимпиадная: содержание тестов не разглашается. На указанном сайте имеется компилятор PascalABC.NET (на данный момент в версии 3.5). Ниже предлагается решение одной из задач с этого Интернет-ресурса.

Задача 4.33
(Взята с //acmp.ru, задача №760. Транспорт).

Время: 1 сек. Память: 16 Мб Сложность: 24%

Все автовладельцы рано или поздно встречаются с инспекторами ГИБДД. Ваша задача состоит в том, чтобы рассчитать минимальное время прохождения автомобилем прямого участка дороги, на котором стоят инспекторы. Известно, что на данном участке инспекторы останавливают все проезжающие мимо автомобили. Примем следующие допущения: автомобили являются материальными точками, которые могут мгновенно останавливаться или набирать максимальную скорость. Автомобиль начинает свой путь в точке 0 и заканчивает его в точке L.

Входные данные
Первая строка входного файла INPUT.TXT содержит три числа: N – количество инспекторов ($0 \le N \le 30$), V – максимальная скорость автомобиля (км/ч) и L – длина участка дороги (км) ($1 \le V, L \le 200$). Вторая строка содержит N пар чисел x_i – расстояние от точки 0 до инспектора ($0 \le x_i \le L$) и t_i – время (мин), на которое он останавливает машину ($1 \le t_i \le 10$). Все инспектора стоят в разных точках. Все числа во входных данных целые.

Выходные данные
В выходной файл OUTPUT.TXT выведите время преодоления участка дороги автомобилем в минутах с точностью до двух знаков после запятой (вещественное число с ровно двумя цифрами после запятой).

Как решать подобную задачу на этом сайте? Прежде всего, фразы «входной файл INPUT.TXT» и «выходной файл OUTPUT.TXT» вы можете полностью игнорировать. Когда пишете программу, вводите данные с клавиатуры и смотрите результаты на экране. Никаких проблем. Далее, написанную программу следует вставить через буфер обмена (Ctrl+A и затем Ctrl+C в среде PascalABC.NET, а далее Ctrl+V на сайте) в соответствующее поле, предварительно выбрав компилятор PascalABC.NET. Если отправите файлом, ваш текст попадет «в руки» компилятору Free Pascal. С соответствующим печальным результатом.

В задаче, как это бывает на олимпиадах, без подвоха не обошлось. Конечно, можно разбить путь на отрезки, зная координаты инспекторов, но зачем? Задана длина пути и постоянная скорость. Мгновенные остановки и разгоны, а также прямой участок дороги позволяют считать движение условно равномерным и прямолинейным на всем протяжении. Следовательно, суммарное время преодоления всех участков может быть найдено как отношение пройденного пути к скорости движения. Для перевода времени в минуты нужно будет умножить его на 60. Остается учесть потери времени на остановки. Они равны сумме времени остановки на каждом участке. Находим эту сумму и прибавляем ее к времени движения. Переменная *p* в программе служит для считывания координат инспекторов, которые в расчете не участвуют. Поэтому предупреждающее сообщение компилятора о том, что переменная *p* получает значение, которое затем не используется нужно игнорировать. С диапазоном представления данных проблем нет, используем обычные **integer** и **real**.

```
begin
  var n := ReadInteger;
  var (V, L) := ReadlnReal2;
  var (S, t) := (60 * L / V, 0);
  loop n do
  begin
    var (p, ti) := ReadInteger2;
    t += ti
  end;
  Write(S + t:0:2)
end.
```

Вот и все. Как видите, совсем несложно решать в PascalABC.NET и задачи олимпиадного уровня.

4.12. Вместо заключения

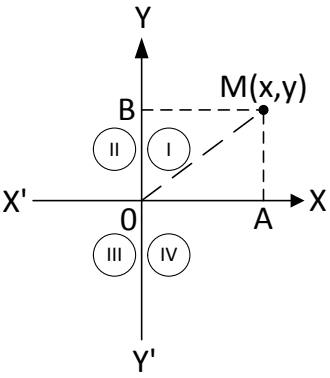
На этом заканчивается материал первой части книги. Во второй части вас ждет знакомство с подпрограммами, последовательностями и массивами. Именно там в полной мере раскрываются все достоинства и преимущества языка PascalABC.NET. До встречи на страницах второй части!

Приложения

В приложениях…

Прямоугольная декартова система координат
Уравнения плоских линий
Формулы из геометрии
Алгоритмы вычисления сумм и произведений
Получение цифр в числе
Схема Горнера для полинома
Чтение программ в КИМ ЕГЭ и ОГЭ

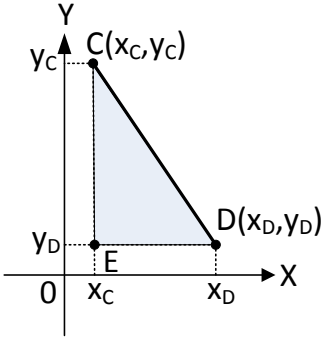
П1. Прямоугольная система координат



Прямоугольная декартова система координат – основная изучаемая в школе. На плоскости она образуется посредством двух взаимно перпендикулярных координатных осей X'X и Y'Y. Точка пересечения этих осей O называется началом координат. Ось X'X (она же – просто ось X) горизонтальна и направлена вправо. Ось Y'Y (она же – ось Y) вертикальна и направлена вверх. Координатные оси разбивают плоскость на четыре четверти, которые нумеруются римскими цифрами от I до IV. Положение произвольной точки M на плоскости определяется двумя координатами x и y, значения которых равны длине отрезков OA и OB (см. рисунок). Координаты точки записываются после ее имени в круглых скобках. Буквы X' и Y' обычно указывать не принято. Если пространство, в котором находится точка, трехмерное, добавляется третья координата z, лежащая на координатной оси Z'Z, расположенной перпендикулярно плоскости с точками O, A и B.

Расстояние от точки M до начала координат можно определить по теореме Пифагора, если соединить ее с началом координат и рассмотреть полученный треугольник OMA.

$OM = \sqrt{OA^2 + AM^2}$ или $L = \sqrt{x^2 + y^2}$



Прямая на плоскости задается двумя точками, принадлежащими этой прямой. В прямоугольной системе координат такая прямая задается координатами пары точек. Например, прямая CD может определяться координатами C(x_C, y_C), D(x_D, y_D). Выполним проекции точек C и D на координатные оси. Пересечение линий проекций обозначим точкой E. Как и в приведенном выше случае,

получился треугольник, который обозначим CDE. Но длины катетов теперь находятся как $x_D - x_C$ и $y_D - y_C$. Длина отрезка прямой определяется по теореме Пифагора

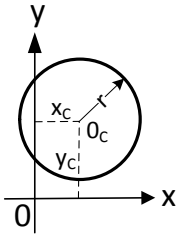
$$L = \sqrt{(x_D - x_C)^2 + (y_D - y_C)^2}$$

Треугольник на плоскости задается координатами своих вершин. Три вершины – шесть координат.

Четырехугольник (а также многоугольники с большим количеством вершин) задаются координатами каждой вершины. Но существуют частные случаи.

Прямоугольник со сторонами, параллельными осям координат, задается координатами своей диагонали, т.е. так же, как прямая.

Окружность задается координатами центра $O_C(x_C, y_C)$ и величиной радиуса r. Также, вместо величины радиуса может задаваться координата любой точки, лежащей на окружности, что позволяет вычислить радиус как расстояние между двумя точками. Если центр окружности совпадает с началом координат, достаточно задать только радиус окружности.



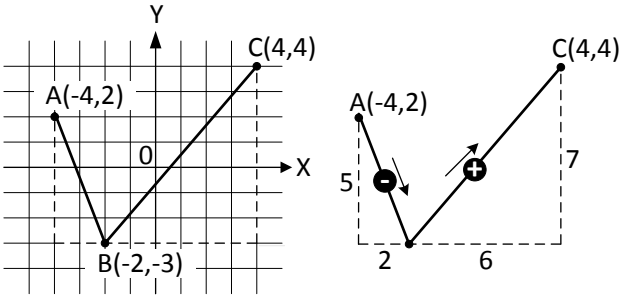
Правильный многоугольник может быть задан количеством вершин, координатами центра описанной окружности и длиной радиуса, либо координатами одной из вершин.

П2. Уравнения плоских линий

Прямая линия, проходящая через две точки $A(x_1,y_1)$ и $B(x_2,y_2)$ может быть описана в виде $y = kx + b$, где коэффициенты k и b находятся по формулам

$$k = \frac{y_2 - y_1}{x_2 - x_1}; \quad b = y_1 - k \cdot x_1 \qquad (П2.1)$$

Чтобы не запоминать эти формулы, можно поступить следующим образом (это мнемоника для нахождения уравнения в информатике, а не объяснение к уроку математики).



Пусть имеется фрагмент кусочно описанной функции, графически изображенной на левом рисунке. Коэффициенты уравнения прямой AB, записанного в виде $y = ax + b$, можно найти по формулам П2.1, если вы их помните. Если нет – попробуйте следующий способ.

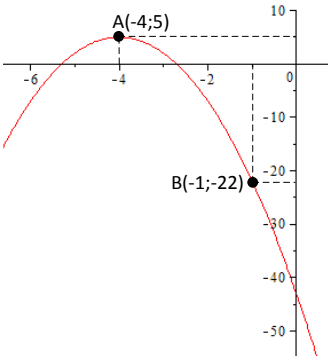
Сначала разыщем значение k. Двигаясь слева направо, мы «спускаемся с горы», высота **понижается**, поэтому знак будет «минус». Мы спустимся на **пять** клеток, переместившись вправо на **две**. Это и даст нам значение $k = -5/2$ или $k = -2{,}5$. Теперь мы можем записать уравнение в виде $y= -2.5x + b$, откуда $b = y - (-2.5x)$. Подставим значения x и y в любой известной точке, например, A и получим $b = 2 - (-2.5)\cdot(-4)$. Несложное вычисление дает значение $b = 12$. Готово. Искомое уравнение $y = -2.5x + 12$.

Аналогичным рассуждением найдем уравнение для прямой BC. Здесь придется «подниматься в гору», высота будет **расти**, поэтому знак k будет «плюс». Поднимемся на 7, переместившись вправо на 6, поэтому $k = 7/6$. Подставляя координаты из точки C в выражение $b = y - 7/6x$, получаем $b = 4 - 7/6 \cdot 4 = -2/3$. Уравнение найдено: $y = 7/6 \cdot x – 2/3$.

Если вы сомневаетесь в правильности полученного решения, подставьте в полученное уравнение координаты второй точки прямой, например, B. Должно получиться тождество.

Квадратная парабола, проходящая через три точки $A(x_1,y_2)$, $B(x_2,y_2)$, $C(x_3,y_3)$ может быть описана в виде $y = ax^2 + bx + c$, где коэффициенты a, b, c находятся по формулам

$$a = \frac{y_3 - \frac{x_3(y_2 - y_1) + x_2 y_1 - x_1 y_2}{x_2 - x_1}}{x_3(x_3 - x_1 - x_2) + x_1 x_2}; \quad b = \frac{y_2 - y_1}{x_2 - x_1} - a(x_1 + x_2); \quad c = \frac{x_2 y_1 - x_1 y_2}{x_2 - x_1} + a x_1 x_2 \qquad (П2.2)$$



Громоздкие формулы, верно же? Запомнить их вообще нереально. Памятуя об этом, составители экзаменационных задач по информатике проявляют известную гуманность и не рисуют участков графиков в виде дуг, принадлежащих лишь одной ветви квадратной параболы. Иными словами, на графике практически всегда присутствует вершина параболы, причем или для нее указываются координаты, или их легко взять из графика «по клеточкам». Наличие вершины параболы все настолько упрощает, что формулы П2.2 становятся излишними.

Нам требуется получить уравнение в виде $y = f(x)$, и совсем необязательно именно в виде $y = ax^2 + bx + c$. В данном случае намного удобнее вид $y = k(x - p)^2 + q$.

Зная координаты вершины параболы $A(x_A,y_A)$, можно сразу записать, что $p = x_A$, $q = y_A$. Вот почему так важно знать координаты вершины! Ведь теперь осталось лишь найти k и для этого используются координаты второй точки. У нас они есть: в точке B x = -1, y = -22. Подставим известные значения в искомое уравнение.
$-22 = k(-1 + 4) + 5$; $3k = -27$; $k = -3$. Окончательно $y = -3(x + 4)^2 + 5$.

Окружность радиуса R, имеющая центр в точке $O(x_C,y_C)$ описывается уравнением вида

$$(x - x_C)^2 + (y - y_C)^2 = R^2 \qquad (П2.3)$$

П3. Геометрия в вычислениях

1. Длина прямой AB, заданной точками A(x_1,y_1) и B(x_2,y_2)

$$L_{AB} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \qquad (П3.1)$$

```
var Lab := Sqrt((Sqr(x2 - x1) + Sqr(y2 – y1));
```

2. Периметр n-угольника, заданного координатами своих вершин, определяется как сумма длин отрезков прямых, соединяющих соседние вершины.

3. Расстояние от точки M(x,y) **до прямой**, проходящей через точки A(x_1,y_1) и B(x_2,y_2)

$$h = \left| \frac{(x_2 - x_1)(y - y_1) - (y_2 - y_1)(x - x_1)}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}} \right| \qquad (П3.2)$$

```
var h := Abs(((x2 - x1) * (y - y1) - (y2 - y1) * (x - x1)) /
      Sqrt((Sqr(x2 - x1) + Sqr(y2 - y1))));
```

Обратите внимание, что формула П3.2 может применяться также для определения кратчайшего расстояния от точки до отрезка, если перпендикуляр, опущенный из точки M на отрезок AB, попадает именно на отрезок, а не на его продолжение.

4. Площадь треугольника с вершинами A(x_A,y_A), B(x_B,y_B), C(x_C,y_C)

$$S = \frac{1}{2}\left|(x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1)\right| \qquad (П3.3)$$

```
var S := 0.5 * Abs((x2 - x1) * (y3 - y1) - (x3 - x1) * (y2 - y1));
```

Можно также воспользоваться формулой Герона, но это приведет к гораздо более длинным вычислениям. При целочисленных координатах результат, полученный с помощью формулы Герона, следует округлить с точностью до 0.5.

П4. Нахождение сумм и произведений

Получение суммы S ряда n чисел a_1, a_2, ... a_n

$$S = \sum_{i=1}^{n} a_i$$

1. Положить S = 0, i = 1
2. Если i > n, завершить алгоритм
3. Увеличить S на a_i
4. Увеличить i на 1
5. Перейти к 2.

Реализация с циклом **for**:

```
var s := 0;
for var i := 1 to n do
begin
  var a := ReadInteger; // как пример
  s += a
end;
```

Реализация с циклом **loop**:

```
var (s, i) := (0, 1);
loop n do
begin
  var a := ReadInteger; // как пример
  s += a;
  i += 1
end;
```

Реализация с циклом **while**:

```
var (s, i) := (0, 1);
while i <= n do
begin
  var a := ReadInteger; // как пример
  s += a;
  i += 1
end;
```

Реализация с циклом **repeat**:

```
var (s, i) := (0, 1);
if i <= n then
  repeat
    var a := ReadInteger; // как пример
    s += a;
    i += 1
  until i > n;
```

Получение произведения П ряда n чисел a_1, a_2, … a_n

$$\Pi = \prod_{i=1}^{n} a_i$$

1. Положить П = 1, i = 1
2. Если i > n, завершить алгоритм
3. Умножить П на a_i
4. Увеличить i на 1
5. Перейти к 2.

Реализация с циклом **for**:

```
var p := 1;  // укажите 1.0 вместо 1, если возможно переполнение
for var i := 1 to n do
begin
  var a := ReadInteger; // как пример
  p *= a
end;
```

Вычисление n! – факториала числа n. Факториал n – произведение натуральных чисел от 1 до n. Вычисляется по алгоритму нахождения произведения для a_i = i. Без вычислений считается, что 0! = 1! = 1. Имеется рекуррентная формула n! = n×(n-1)!

$$n! = \prod_{i=1}^{n} i$$

1. Положить П = 1, i = 1
2. Если i > n, завершить алгоритм
3. Умножить П на i
4. Увеличить i на 1
5. Перейти к 2.

Реализация с циклом **for**:

```
var p := 1;  // если n < 13, иначе – неверный результат!
for var i := 1 to n do
  p *= i;
```

Если n от 13 до 20:

```
var p := int64(1);
for var i := 1 to n do
  p *= i;
```

Если n < 171 (точны только первые 15 знаков):

```
var p := 1.0;  //
for var i := 1 to n do
  p *= i;
```

Для любого n точны все знаки:

```
var p := BigInteger.One;
for var i := 1 to n do
  p *= i;
```

П5. Получение цифр в числе

Десятичное натуральное число n из k цифр в расширенном виде по степеням основания 10 можно записать как
$n = a_{k-1} \cdot 10^{k-1} + a_{k-2} \cdot 10^{k-2} + \ldots + a_1 \cdot 10^1 + a_0 \cdot 10^0$, где a_{k-1}, a_{k-2}, … a_0 – цифры числа.

Для выделения первой цифры числа нужно нацело разделить n на 10^{k-1}. Последнюю цифру получаем как остаток от целочисленного деления n на 10. С остальными цифрами несколько сложнее. Имеются два алгоритма выделения нужной цифры. Первый предполагает, что сначала при помощи получения остатка от деления (**mod**) отбрасываются все цифры слева от искомой, а затем при помощи деления (**div**) отбрасываются все цифры справа от искомой. Второй алгоритм повторяет первый с точностью до наоборот. Вначале отбрасываются цифры справа от искомой, а затем – слева.

Рассмотрим пример. Требуется получить вторую слева цифру в пятизначном числе. Пусть n = 52091 и нужно получить цифру 2. Используем первый алгоритм. Вторая слева в пятизначном числе – это четвертая справа. Отбрасываем цифры левее четвертой, для чего находим остаток от деления n на 10^4. 52091 **mod** 10000 дает в результате 2091. Делим результат на 10^3 (степень на единицу меньшая, чем в предыдущей операции). 2091 **div** 1000 даст результат 2.

Мнемоническое правило: Чтобы найти m-ю справа цифру в числе, находим остаток от деления этого числа на величину, записываемую как единица с m нулями, а затем делим результат на число, в десять раз меньшее.

В PascalABC.NET операции **mod** и **div** имеют равный приоритет, поэтому решение приведенного выше примера можно записать как *n* ***mod*** *10000* ***div*** *1000*, но если в таком виде имеются трудности с восприятием, пишите *(n* ***mod*** *10000)* ***div*** *1000*.

Для получения всех цифр числа *n* с заранее неизвестным количеством цифр, используется следующий алгоритм (он дает цифры числа *d* в обратном порядке):
1. Если *n* = 0, закончить алгоритм
2. Получить очередную цифру числа *d* как остаток от деления *n* на 10
3. Принять в качестве нового значения *n* целую часть от деления *n* на 10
4. Перейти к 1.

Наиболее популярная реализация этого алгоритма – цикл **while**.

```
while n > 0 do
begin
  var d := n mod 10;
  // тут что-то делаем с полученной цифрой d
  n := n div 10;
end;
```

П6. Схема Горнера для полинома

Полином P(x) степени n в общем виде может быть записан как
$P(x) = a_0x^n + a_1x^{n-1} + \ldots + a_{n-1}x^1 + a_nx^0 = a_0x^n + a_1x^{n-1} + \ldots + a_{n-1}x + a_n$.

Чтобы избежать использования операции возведения в степень и при этом минимизировать количество операций умножения, полином записывают в эквивалентном виде, называемом схемой Горнера
$P(x) = a_n + x(a_{n-1} + x(a_{n-2} + \ldots + x(a_1 + a_0x)\ldots))$

Обратите внимание, что везде стоят знаки плюс, а коэффициенты берутся со своим знаком. Так удобнее, чтобы не ошибиться. Во второй книге, где будут рассмотрены последовательности и массивы, станут понятны все преимущества схемы Горнера, а пока полезно привыкнуть пользоваться ей, записывая коэффициенты непосредственно.

Рассмотрим пример. $P(x) = 6.5x^3 - 4x^2 + 0.12x + 12.8$. По схеме Горнера записываем $12.8+x(0.12+x(-4+6.5x))$.

Схема несложна, можно воспользоваться следующим алгоритмом: записываем коэффициент с его знаком, начиная со свободного члена, далее ставим знак плюс и пишем аргумент (в нашем случае x). Затем открываем скобку и повторяем действия для следующего коэффициента. Для коэффициента при старшей степени скобку не открываем, а сразу умножаем его на аргумент. Затем закрываем столько скобок, сколько открыли. Вы всегда можете проверить правильность записи, раскрыв скобки, ведь это обычная алгебра.

Если какие-то степени в полиноме не представлены, соответствующие коэффициенты полагаем равными нулю. Если нулевых коэффициентов много, имеет смысл подумать об ином способе вычисления, выполнив какие-либо преобразования.

$P(x) = x^8 + 3x^5 – 0.4x + 7.2 = x^3(1 + 3x^2) – 0.4x + 7.2$

```
var P := x * x * x * (1 + 3 * x * x) - 0.4 * x + 7.2;
```

В данном случае короче и проще будет написать именно так, а не по схеме Горнера. Для восьмой степени пришлось бы писать довольно много членов, умножая их на ноль. Фанатизм по отношению к схеме Горнера проявлять не нужно: для высоких степеней она записывается кратко только при использовании массивов или последовательностей.

П7. Чтение программ в КИМ ЕГЭ и ОГЭ

В КИМ ЕГЭ и ОГЭ тексты программ на Паскале приводятся в самой примитивной его версии, Turbo Pascal (далее – TP). Компилятор TP был выпущен в 1983 году и просуществовал 20 лет, после чего был вытеснен Borland Delphi. Все прочие компиляторы и интерпретаторы, использующие Паскаль, считаются в рамках изучаемого в школах подмножества языка совместимыми с TP.

Умея программировать на PascalABC.NET, нужно всего лишь понять, какие конструкции и способы записи в языке TP устарели и не применяются при написании программ.

Типичная программа в TP имеет следующую структуру

```
program Vasya; { заголовок программы }
const описание констант; { раздел описания констант }
var описание переменных; { раздел описания переменных }
begin
    тело программы
end.
```

Как вы наверно уже догадались, в фигурных скобках пишут *комментарии.* Те самые, которые мы указывали после //. Понятно, что такие скобочки вы тоже можете писать в своих программах, поскольку PascalABC.NET понимает коды TP.

Заголовок ***program***. Как теперь говорят – «ни о чём». Анахронизм, рудимент, пережиток прошлого. Забудьте.

Раздел описания констант ***const***. Мы о константах пока не говорили, потому что нужды не было. Константа – она как переменная, только ее значение нельзя менять. Компилятор не разрешит. Описание – как у переменной **var** с инициализацией, только вместо знака присваивания := указывается =

```
const g = 9.81; pi = 3.14; n: integer = 10;
```

Если задание предполагает поиск ошибок, ваша задача – отследить, нет ли попыток присвоить константе значение, для чего ее имя должно появиться в левой части оператора присваивания.

Раздел описания переменных ***var***. Это обычные объявления типов переменных. Те, которые вы делали. Только писали их по мере надобности и непосредственно там, где это требовалось. В ТР предполагалось, что перед написанием программы вы садитесь и долго думаете, какие переменные взять и какой тип им назначить. Тут были полезны блок-схемы, для чего их и рисовали. В PascalABC.NET мы разобрали 70 задач – вы там встретили хоть одну блок-схему? Нет, потому что в них не было нужды. Ну и еще одно отличие: в ТР при описании переменных инициализировать их нельзя. Т.е. описываем в разделе описания, а инициализируем уже в программе.

Типы данных. Тут несколько иначе, чем в PascalABC.NET.

Вещественный тип **single** – это такое забавное четырехбайтное недоразумение с семью значащими цифрами и диапазоном до 10^{38}. Нормальное вещественное – это **double**, ему и соответствует **real** в PascalABC.NET. Конечно же, для понимания программы это ничего не меняет - вещественное – оно и есть вещественное. Не целое – вот и все.

С целыми тоже нагородили. Однобайтный тип **shortint** вмещает числа от -128 до 127. Однобайтный тип **byte** – для чисел от 0 до 255. Два байта **word** позволяют записать числа от 0 до 65535. Двухбайтный (!) **integer** отведен для диапазона от -32768 до 32767. И, наконец, четырехбайтный **longint** – это и есть привычное вам **integer** из PascalABC.NET.

> В PascalABC.NET типу **integer** ТР соответствует **smallint**, а типу **longint** ТР – тип **integer**.

Вся эта чехарда с разными типами переменных возникла во времена, когда мегабайт памяти стоил дороже, чем сейчас автомобиль. Но вообще-то в ЕГЭ этой ерунды с обилием типов вроде пока нет.

Имена переменных. Последовательность латинских букв, цифр и знака подчеркивания, начинающаяся не цифрой. Вот так все строго.

Тело программы. Это блок **begin** … **end**, завершающийся точкой. Ничего не поменялось.

Набор функций у ТР существенно беднее, поэтому не удивляйтесь, если встретите Sin(x) / Cos(x) – это всего лишь тангенс, потому что функции Tan(x) в ТР нет. Нет в ТР и ** – операции возведения в степень. Если нужно записать a^b, вспоминают основное логарифмическое тождество $a^b = e^{b \cdot \ln a}$ и пишут Exp(b * Ln(a))

Из *операций* в ТР отсутствуют +=, -=, *= и /=.

Собственно, это все. Я вполне серьезно в начале книги писал, что полчаса достаточно, чтобы научиться читать программы, написанные в ТР.

В предоставленной ФИПИ демоверсии КИМ ЕГЭ-2021 имеются всего два задания (с номерами 6 и 22), в которых требуется читать «старый» код. Приятно, что для его понимания достаточно запомнить только два момента:
- переменные описываются выше основной программы;
- данные **integer** имеют диапазон [-32768 .. 32757].

А зачем тогда столько информации по чтению «древнего кода»? Для того, чтобы вы могли понимать программы из различных учебников и видеоматериалов, которые почему-то все еще часто приводятся на языке Turbo Pascal.