
Современное программирование с нуля!
В. Потопахин
ДМК ПРЕСС
издательство

УДК 32.973.26-018.2
ББК 004.438
П64

Потопахин В.
П64 Современное программирование с нуля! – М.: ДМК Пресс, 2016. – 240 с.: ил.

ISBN 978-5-97060-405-2

Эта книга для тех, кто хочет получить хорошие навыки программирования с использованием языка Компонентный Паскаль -современной версии языка Паскаль. Изложение сопровождается большим количеством примеров, способствующих успешному усвоению материала людьми с различным уровнем подготовки – необходимо только желание и терпение.

Материал курса представлен в виде последовательности прикладных задач, нацеленных на формирование у обучаемого особой программисткой логики и дающих возможность изучить и отработать на практике все существенные особенности языка Компонентный Паскаль.

УДК 32.973.26-018.2
ББК 004.438

Потопахин Виталий Валерьевич

Современное программирование с нуля!

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Корректор *Синяева Г. И.*
Верстка *Старцевой Е. М.*
Дизайн обложки *Харевская И. В.*

Гарнитура «Петербург». Печать офсетная.
Усл. печ. л. 36. Тираж 100 экз.

Web-сайт издательства: www.dmkpress.com

ISBN 978-5-97060-405-2

Содержание

Предисловие 4

Глава 1. Неформальное введение 5
Кратко о главном 6
Условные циклы 12
Общая структура программы на КП 16
Условный оператор 21
Какие еще есть типы данных в КП 29
Массивы 30
Вложенные циклы 36
Многомерные массивы 49
Процедуры 57
Рекурсия 75
Записи 86
Указательные типы 94
Связные списки 96
Деревья 106
Файлы 114

Глава 2. Систематическое введение в КП 117
Введение 118
Понятие числа 127
Понятие идентификатора 127
Величины. Типы данных. Объявление и виды типов 130
Операции 142
Операторы 145
Модули 160
Полный список предопределенных процедур 161

Глава 3. Практикум 165
Раздел A. Разные задачи 181
Раздел B. Сортировки 198
Раздел C. Задачи перебора 201
Раздел Д. Графы 209

Приложение. Кратко о теории графов 226

Заключение 240

Введение

Приступать к изучению данной главы не рекомендуется без проработки первой. Если же материал неформального введение вами усвоен, то видимо язык КП вы понимаете неплохо и все что нужно это дополнить ваши знания языка и немного их систематизировать. Идеально систематизирована информация о языке в сообщении о языке. Сообщение является составной частью документации прилагающейся к среде программирования и принципиально его вполне достаточно. Но сообщение о языке – это очень формализованный текст читать который без специальных навыков достаточно затруднительно. Поэтому вторая глава по своему содержанию и форме представляет собой развернутое сообщение о языке. Определения языка даны средствами формализма Бэкуса-Наура и имеют дополнительные пояснения, дано несколько больше примеров, чем это есть в сообщении. Имеется дополнительная информация, позволяющая лучше понять конструкции языка и его идейную основу. В основном структура сообщения сохранена, но есть некоторые отступления, которые по мнению автора помогут лучше разобраться в тексте. Глава не имеет никаких вопросов для самопроверки и задач для практикума. Этой цели посвящена третья глава – целиком представляющая собой практикум по программированию.

Общие вопросы

Главная проблема общения между человеком и компьютером это огромный смысловой разрыв между естественным языком человека и языком машины. Вот некоторые из различий:

- в естественном языке огромный набор понятий, компьютер использует крайне ограниченный и даже скудный понятийный аппарат;
- естественный язык отражает целое множество мыслительных инструментов используемых человеческим интеллектом. Человек, может обобщать, абстрагировать и т.д. и т.д. Компьютер способен только к воспроизведению алгоритмов;
- человеческие понятия многозначны, их конкретное наполнение зависит от различных контекстов, от культуры и степени развития конкретного человека. Машинные понятия однозначны и практически не изменяются при переходе от машины к машине. Нельзя сказать, что совсем не меняются, но эти изменения не столь значительны.

Можно привести и другие различия, но даже сказанного достаточно, чтобы понять, между человеческим изложением решения задачи и машинно-пригодным находится огромная пропасть. С одной стороны алгоритм, записанный на естественном языке невозможно подвергнуть компиляции (переводу в машинный код), с другой стороны алгоритм, записанный на языке машинных кодов очень труден для понимания.

Выход из положения был найден в виде языка посредника. Такой язык опирается на небольшое количество базовых, строго определенных понятий (однозначно

понимаемых), смысл которых достаточно близок к понятийному аппарату используемому человеком. Такие языки были названы языками высокого уровня.

С появлением языков высокого уровня программирование, как вид деятельности не только стало возможным для большого количества специалистов, работающих в разных прикладных областях, но и дало большие преимущества для профессиональных программистов. Впрочем, можно сказать, что с появлением уже первых языков термин «профессиональный программист» стал в значительной степени размываться.

О борьбе с ошибками

Язык высокого уровня не смотря на высокий уровень строгости не понятен компьютеру и нуждается в переводе на машинный язык. Это в свою очередь создает необходимость разработки специальных программ – трансляторов обеспечивающих возможность такого перевода.

Появление языков высокого уровня и их трансляторов вызвало к жизни важный вопрос – как бороться с ошибками программиста. Вопрос этот конечно стоял всегда, но в эпоху программирования в машинных кодах, ответ на него давался автоматически: все что делает программа лежит на совести программиста. Появление трансляторов ответ на этот вопрос усложнило. Транслятор, конечно не участвует в разработке алгоритма, но фактически участвует в написании программы. Он занимается ее переводом, и в процессе перевода выполняет анализ текста, а значит может обнаруживать какие-то ошибки. Время затрачиваемое программистом на борьбу с ошибками сопоставимо со временем разработки программы, а зачастую и превышает его, поэтому возможность автоматизации поиска ошибок безусловно очень важна.

И вот здесь оказалось, что для минимизации ошибок, далеко не все равно, как устроен язык. Чем больше предоставляет язык возможностей программисту, тем больше программист может совершить ошибок и тем сложнее их будет обнаружить. Поэтому проектирование языка это поиск золотой середины между простотой и ясностью с одной стороны и обилием возможностей с другой.

Существуют различные точки зрения о том, где находится эта золотая середина. Мы же будем придерживаться того мнения, что потери времени на исправление ошибок дороже самых широких возможностей. И язык программирования должен предоставлять лишь то, что является жизненно важным. Язык это базовый минимум, позволяющий писать кристально ясные, хорошо читаемые программы.

Кстати такой подход совершенно не противоречит идее больших возможностей. Надо просто различать две различные сущности: язык программирования и среду программирования, которая может содержать многочисленные расширения языка и обеспечивать любой уровень сложности.

Язык и определение алгоритма

Программирование в своей основе опирается на понятие алгоритма, которое имеет строгое определение, но так уж получилось, что не одно. Поэтому для

разработчика языка прежде всего необходимо решить вопрос, на какое представление о алгоритме он будет опираться. Мы не будем сейчас уходить глубоко в теорию алгоритмов, отметим только, что существуют два основных подхода:

- декларативное программирование;
- императивное программирование.

При декларативном походе программа (алгоритм) понимается, как некая система определений того, что должно получится. При императивном подходе программа понимается, как последовательность действий, выполнение которых некоторым исполнителем приводит однозначно, к требуемому результату.

Наш выбор – императивное программирование. Возможно понимание алгоритма, как последовательности действий наиболее близко человеческому интеллекту. Это так по крайней мере с точки зрения автора этого текста, этой точки зрения придерживаются многие намного более авторитетные люди в области программирования, но конечно специалисты по декларативным языкам найдут аргументы в пользу своей точки зрения.

Минимальный набор действий

Из сказанного выше вытекает задача определения минимально необходимого набора действий. Заметим, что на уровне процессора все происходящее сводится к преобразованиям чисел. Все, что мы на высоком уровне можем делать с числами, записывается арифметическими выражениями вида:

```
Результат = Выражение
```

Это первая базовая языковая возможность. Она называется «ПРИСВАИВАНИЕ». Ее смысл в вычислении выражения записанного справа от равенства и присвоение полученного выражения величине, чье имя указано слева от равенства.

Может возникнуть потребность выполнить некоторую последовательность действий многократно, без многократной их записи. Соответствующая языковая конструкция называется циклом.

Последовательность выполняемых действий может разветвляться в зависимости от результата вычисления некоторых условий. Для организации ветвлений язык предоставляет условную конструкцию. Графически цикл и ветвление можно представить схематически.

Блок-схема цикла (рис 2.1) читается так: пока истинно условие выполняется последовательность действий, если условие ложно, управление передается на команду следующую за циклом.

Блок-схема ветвления (рис. 2.2) читается так: выбор исполняемой последовательности действий происходит в зависимости от истинности условия. После исполнения выбранной последовательности управление передается на команду следующую за ветвлением.

Конечно, это самые общие конструкции, их реализация в языке высокого уровня может быть различной, даже более того, в одном и том же языке, успешно сосуществуют различные реализации, имеющие отличные друг от друга свойства и особенности.

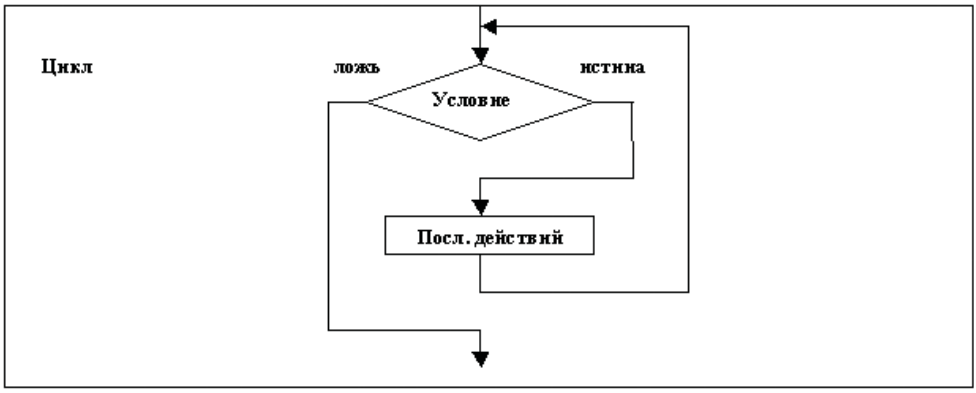

Рис. 2.1. Конструкция цикла

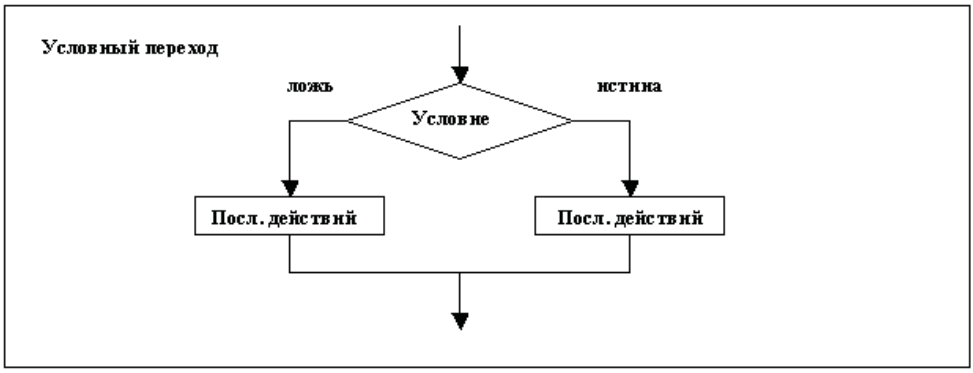

Рис. 2.2. Ветвления

Следующая важная языковая конструкция это процедура (функция, подпрограмма). Процедура – это в некотором смысле программа. Во всяком случае она обладает всеми свойствами программы. Она реализует собственный алгоритм, оформляется отдельно от всех прочих текстов (других процедур, модулей и т.д). Смысл процедур в поддержке процесса декомпозиции задачи. Если задача достаточно велика, при проектировании решения задача разбивается на логически независимые подзадачи, каждая из которых оформляется в виде отдельного решения – процедуры и уже затем из процедур, как из кирпичиков собирается полное законченное решение большой задачи.

Отношение КП к набору операций. Компонентный Паскаль построен по принципу минимальности набора действий. Это не в ущерб функциональности. В языке присутствуют три типа цикла, два типа условного оператора, набор достаточный для полноценно функционирующего языка программирования.

Типы данных

Язык определяется двумя сущностями: набором выполняемых операций и набором структур данных. Данное (величина) определяется именем, значением и типом. Имя дается программистом, на имя не возлагается никакого серьезного функционального смысла, значение величины определяется логикой программы, в процессе ее работы. Наиболее значимо для величины понятие типа. Тип данных определяет множество значений, которое может принимать величина и набор операций, которые над этой величиной можно выполнять.

Структуры данных любого языка программирования бывают двух типов: основные и составные. Основные типы это наиболее простые. Это например различного вида числа и литеры. Составные это массивы – упорядоченные множества однотипных данных и записи – множества данных различного типа. Кроме того, данные делятся на статические – память под которые выделяется на этапе компиляции и динамические, процессом создания и удаления которых можно управлять во время работы программы.

Виды типизации

Вернемся к вопросу о программных ошибках. Значительная их часть сводится к неправильному использованию данных. Состояние ошибки возникает если некая операция использует данные неразрешенного типа. Нет большого смысла складывать литеры, например так:

```
Число=Литера + Литера
```

Такую операцию есть смысл запретить. Наверное можно запретить присваивания в результате которого число занимающее много памяти присваивается переменной под которую выделено мало памяти. В общем незаконными являются операции передачи значений от переменной (выражения) одного типа к переменной другого типа, если ***типы переменных (выражений) не соответствуют друг другу по размеру выделенной памяти и набору допустимых операций***.

Вопрос соответствия в разных языках программирования разрешается по разному, но все возможные варианты сводятся к двум терминам: «сильная типизация» и «слабая типизация».

Слабая типизация. Возможно почти все. Значение любого типа можно передать почти любой величине. Степень этого «почти» в разных языках различна. Не совпадающие типы величин при передаче автоматически или как еще говорят «неявно» преобразуются. Программист должен хорошо понимать правила преобразования и он сам несет ответственность за результат. Языки со слабой типизацией дают большую свободу в обращении с памятью и величинами, но платить за это приходится высокой вероятностью трудно обнаруживаемых ошибок.

Сильная типизация. Передача значения возможна только при совместимости типов. Например, целое можно присвоить целому. В этом случае мы имеем полное совпадение типов. Целое можно присвоить вещественному. Здесь типы не совпадают, но они совместимы, в том смысле, что целое значение можно разместить

в области памяти выделенной под вещественную величину и они совместимы по арифметическим операциям. Вещественное же целому присвоить нельзя, так как под вещественное число требуется больше памяти. Нельзя присвоить литеру никакому числовому типу. Нельзя выполнить присвоение двух записей с разным набором полей. Сильная типизация делится на два вида:

- *сильная структурная типизация*. Два типа считаются одинаковыми, если они совпадают с точностью до структуры. Говорить о таком виде типизации можно разумеется только в отношении типов обладающих структурой (записей). Две записи считаются одинаковыми если они имеют одинаковый набор полей, при этом имена их типов могут различаться;
- *сильная именная типизация*. Две переменных считаются одинакового типа, если совпадают имена их типов. Легко понять, что именная типизация накладывает более жесткие ограничения. Записи имеющие одинаковый тип с точки зрения именной типизации очевидно будут совпадать и с точностью до структуры, обратное же неверно.

Отношение КП к типизации. Компонентный Паскаль является языком сильной именной типизации. Для базовых типов вводится совместимость, то есть присвоение допустимо не только для величин одинаковых типов.

Управление памятью. Сборка мусора

Управление памятью заключается в двух действиях выполняемых по ходу работы программы: выделение памяти под структуры данных и возращение памяти в свободную область. Для статических переменных все вопросы с выделением памяти решаются на этапе компиляции. Для динамических величин при необходимости их использования вызывается специальная процедура выделяющая необходимый объем памяти. Более интересен вопрос возврата памяти в свободную область (для этого действия есть специальный термин «сборка мусора»). Здесь две возможности:

Ответственность за удаление ненужных данных возлагается на программиста. В этом случае в языке предусматривается специальная процедура, вызов которой означает уничтожение структуры данных. Предполагается, что программист не ошибется в оценке ситуации и применит процедуру к ненужным данным в правильной точке.

Конечно же это слишком сильное предположение. Не ошибающихся программистов не бывает, это во-первых, а во-вторых, вопрос о ненужности данных можно решить автоматически, при условии, что точно определено, что значит фраза «структура данных не нужна».

Для КП структура данных считается подлежащей удалению, если нет ни одного указателя (переменная специального вида) содержащего адрес области памяти, в которой хранится структура. Действительно, если нет ни одного указателя на данное, то у программиста просто нет возможности к данному обратится и следовательно разумно принять решение о освобождении памяти, даже если эта ситуация возникла вследствие программисткой ошибки.

Таким образом, для высвобождения данных достаточно всем указателям связанным с не нужными данными присвоить специальное значение NIL (адрес в никуда). Освобождение памяти также произойдет если всем указателям связанным с данным будут присвоены какие-либо иные адреса (не NIL).

Формальные грамматики. Формализм Бэкуса-Наура

Программа есть осмысленное предложение записанное на специальном языке. Смысл текста программы определяется целями программиста и формулировкой задачи, то есть является внешним по отношению к языку. Поэтому предложение записанное на языке программирования может иметь смысл, но это не является его обязательной характеристикой. Внутренней, обязательной характеристикой является соответствие набору правил, описывающих, что является правильным предложением вне зависимости от его смысла. Набор таких правил называется синтаксисом. Следовательно, описание языка программирования есть описание его синтаксиса.

Правила синтаксиса можно описывать неформально. Например, допустимо следующее правило:

Описание цикла с шагом начинается с ключевого слова **FOR**

Или

За ключевым словом ***VAR*** следует блок описания переменных

Данные правила действительно описывают некий синтаксис, но они не точны, неоднозначны и не решают главной задачи построения системы синтаксических правил.

А для построения языка программирования требуется, чтобы синтаксис был описан на определенном строгом языке. То есть ситуация точно такая же, как с описанием алгоритма. Алгоритм является однозначно понимаемым текстом, поэтому для его записи нужен специальный язык, называемый языком программирования. Язык программирования описывается системой синтаксических правил. Каждое такое правило является однозначно понимаемым текстом, поэтому для его написания нужен опять специальный язык называемый формальной грамматикой. Существует два типа формальных грамматик:

Порождающая грамматика. Порождающая грамматика представляет собой алгоритм позволяющий из некоторого минимального набора предложений построить все допустимые предложения данного языка.

Распознающая грамматика. Распознающая грамматика представляет собой алгоритм проверки текста, позволяющий за конечное число шагов, выяснить является ли текст программой на языке описываемом данной системой правил.

Везде, далее, говоря о формальной грамматике, будем иметь ввиду порождающую грамматику. Три важнейших понятия грамматики это терминальный и нетерминальный символы и лексема.

- **терминал (терминальный символ)** – объект, непосредственно присутствующий в словах языка, соответствующего грамматике, и имеющий конкретное, неизменяемое значение;
- **нетерминал (нетерминальный символ)** – объект, обозначающий какую-либо сущность языка (формулу, выражение и т.д.) и не имеющий конкретного символьного значения;
- **лексема** – последовательность символов, ограниченная специальными символами, например пробелами. И терминальный и нетерминальный символы суть лексемы.

Предложениями языка, заданного грамматикой, являются все последовательности терминалов, выводимые (порождаемые) из начального нетерминала по правилам вывода. Таким образом грамматика языка – это множество терминальных и нетерминальных символов и множество правил вывода. Для описания синтаксиса языка компонентный Паскаль используется так называемый расширенный формализм Бэкуса-Наура (РФБН).

РФБН – это следующий набор правил: альтернативы разделяются символом |. Квадратные скобки [и] означают необязательность заключенного в них выражения, а фигурные скобки { и } означают его возможное повторение (0 или более раз). В случае необходимости для группирования лексем используются круглые скобки (и). Нетерминальные лексемы начинаются с большой буквы (например, Statement). Терминальные лексемы либо начинаются с маленькой буквы (например, ident), либо записаны только большими буквами (например, BEGIN), либо обозначаются цепочками литер (например, «:=»).

Основные термины

Любой язык программирования предполагает небольшой набор основных понятий через которые разворачиваются все языковые смыслы. Перечислим эти понятия (порядок перечисления ни в коем случае не характеризует значимости): *идентификатор, операция, операнд, оператор, переменная, константа, выражение, тип, локальный, глобальный, экспорт, импорт.*

Идентификатор – уникальное имя программного блока (процедуры или модуля) или величины.

Операция – арифметические или логические операции над данными, или операции над символьными цепочками.

Операнд – Аргумент операции. Грамматическая конструкция, обозначающая выражение, задающее значение аргумента операции, иногда операндом называют место, позицию в тексте, где должен стоять аргумент операции.

Оператор – обозначение действия. Операторы различают элементарные и структурированные. Отличие элементарного от структурированного в том, что элементарный оператор не содержит частей, которые сами являются операторами.

Переменная – Величина, чье значение может быть изменено в процессе работы программы и следовательно определяемое в ходе исполнения программы.

Константа – Величина чье значение не может изменяться в ходе работы программы и следовательно определяемое на этапе компиляции.

Выражение – конструкция, описывающая вычислительные правила, в соответствии с которыми комбинируются константы и текущие значения переменных для вычисления других значений посредством применения операций и процедур-функций. Выражения состоят из операндов и операций. Круглые скобки могут использоваться для выражения конкретных связей между операциями и операндами.

Тип – Описание данного. Содержит информацию, позволяющую определить объем памяти, для хранения величины данного типа, набор допустимых над данным операций и совместимость с другими типами данных.

Локальный – понятие используется для ограничения области использования имени. Локальность имени (например переменной) означает, что структурой с данным именем можно пользоваться только в пределах процедуры в которой было дано объявление.

Глобальный – понятие обратное понятию «локальный», означает что структура связанная с именем известна в пределах всего модуля.

Экспорт – все программные конструкции и структуры данных и процедуры определяются в пределах модуля и по умолчанию за пределами модуля не видны. Операция экспорта позволяет передать имя программной конструкции или структуры данных за границы модуля.

Импорт – операция обратная экспорту. Импорт операция позволяющая получить информацию о программной конструкции или структуре данных из другого модуля.

Главное понятие ***программу*** через описанные выше основные понятия можно определить следующим образом: программа это последовательность операторов управляющих вычислением выражений и присвоением полученных значений переменным.

Построение предложений

Программа – это предложение написанное на языке Компонентный Паскаль. Предложение состоит из слов называемых лексемами. Лексема это последовательность символов словаря разделенных пробелами. Лексема может быть:

- идентификатором (именем чего либо);
- числом;
- операцией;
- ограничителем. Понятие ограничителя необходимо для доопределения смысла текста. Ограничителем может быть скобка, ключевое слово языка и т.д.

Пробелы не являются разграничителями лексем в двух случаях: если они появляются внутри литерных цепочек и если они находятся внутри комментариев. Комментарием, то есть текстом не являющимся текстом программы, считается текст записанный между парой литер «(*» и парой литер «*)».

И последнее. Большие и маленькие буквы в словаре КП считаются разными.

Понятие числа

Числа в КП могут быть записаны в двух системах счисления: десятичной и шестнадцатиричной. Если число записано с суффиксом «H» или «L» то это шестнадцатиричное число иначе десятичное. Суффикс «L» предназначен для обозначения 64-х битных констант. Вещественные числа всегда содержат в своей записи десятичную точку. Число 3 будет воспринято как целое. Для обозначения вещественного числа необходимо использовать запись 3.0. Вещественное число может содержать масштабный множитель E. Тогда запись числа распадается на мантиссу и порядок. Мантисса – это последовательность цифр до масштабного множителя E. Порядок это знак и последовательность цифр после E. Если знак отсутствует, то порядок считается положительным. Число 1.2E2 читается как вещественное число 120. Для записи шестнадцатиричных чисел допустимы следующие знаки в качестве цифр: «A», «B», «C», «D», «E», «F».

Понятие идентификатора

Программу, в некотором смысле можно определить, как набор идентификаторов (имен), для каждого из которых задано описание (смысл) и область видимости (блок программы в котором идентификатором можно пользоваться).

Сообщение о языке дает следующее определение идентификатора

```
ident = (letter | "_") {letter | "_" | digit}.
letter = "A" .. "Z" | "a" .. "z" | "А".."Ц" | "Ш".."ц" | "ш".."я".
digit  = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".
```

Примечание. Данное определение безусловно работает, если вы используете среду BlackBox. Если ваша рабочая среда иная, то возможно там символы кириллицы запрещены для построения идентификаторов. Нужно заметить, что для языков программирования более типична ориентация на символы латинского алфавита.

Построим имя идентификатора пользуясь данным определением. Понятие идентификатора здесь выражается термином ident. Первая круглая скобка говорит о том, что идентификатор может начинаться с letter либо с символа подчеркивания. Далее дано определение нетерминала letter. Это может быть строчная, либо прописная буква латинского алфавита, строчная, либо прописная буква кириллицы.

Следовательно, мы можем выбрать в качестве первого символа идентификатора любую букву, либо символ подчеркивания. Далее, в определении находятся фигурные скобки, в которых записаны три альтернативы: нетерминал letter, определение которого уже рассмотрено, символ подчеркивания и нетерминал digit, представляющий собой один из цифровых символов. Таким образом после первого символа возможно многократное повторение букв, цифр и подчеркиваний. Примеры правильных идентификаторов:

```
A1, ADCD, A_123, df34_8
```

Заметим, из определения следует, что идентификатор не может начинаться с цифры. Эту возможность запрещает первое правило, четко утверждающее, что

первый символ это буква, либо подчеркивание, а использование цифр возможно только со второй позиции.

Правила объявления и видимости идентификаторов:

Идентификаторы используются для обозначения различных объектов: констант, типов, переменных, процедур. Идентификатор должен быть объявлен с указанием типа для переменных величин и для каждого идентификатора определяется область видимости, то есть блок программы в котором данным идентификатором можно пользоваться. Программный блок – это модуль, процедура, запись. Поведение идентификатора по отношению к области видимости описывают четыре правила:

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

Неверно:

Таблица 2.1. Примеры ошибочного объявления

VAR	VAR	VAR
A, b:INTEGER;	A:INTEGER;	A: REAL;
A:REAL;	A, B:INTEGER;	A: ARRAY 10 OF INTEGER;

Локальный характер использования. На объект можно сослаться только в его области видимости;

Неверно:

```
PROCEDURE Example1;
VAR
    A:INTEGER;
BEGIN
END Example1;
PROCEDURE Example2;
BEGIN
A:=1; (*Область видимости величины A это процедура Example1*)
END Example2;
```

Правило определения идентификаторов через другие идентификаторы. Описание типа T, содержащее ссылки на другой тип T1 могут стоять в точках, где T1 еще не известен. Но тогда описание типа T1 должно следовать далее в том же блоке, в котором локализован T;

Пример:

```
TYPE
    Mas=A;
    A=ARRAY 10 OF INTEGER;
```

В определении на момент объявления идентификатора Mas идентификатор A не описан, но описание A находится с описанием Mas в одном программном блоке, поэтому здесь нет ошибки.

Следующий пример:

```
PROCEDURE P1;
TYPE
    Mas=A;
PROCEDURE P2;
TYPE
    A=ARRAY 10 OF INTEGER;
END P2;
BEGIN
END P1;
```

Представляет собой более сложную ситуацию. Можно рассматривать два идентификатора A и Mas как объявленные в одном программном блоке – процедуре P2. С этой точки зрения правила объявления выполнены, но такая точка зрения неверна, для компилятора процедуры P1 и P2 разные программные блоки, даже несмотря на то, что P12 вложена в P1. Небольшая перестановка объявлений устраняет ошибку:

```
PROCEDURE P1;
TYPE
A=ARRAY 10 OF INTEGER;
PROCEDURE P2;
TYPE
Mas=A;
END P2;
BEGIN
END P1;
```

В таком варианте компилятор не обнаружит ошибки. Идентификаторы A и Mas по прежнему расположены в разных программных блоках, но сейчас определяющий идентификатор A по тексту предшествует определяемому Mas в объемлющей процедуре. То что процедура P1 является объемлющей момент принципиальный. Следующий пример демонстрирует почему:

```
PROCEDURE P1;
TYPE
A=ARRAY 10 OF INTEGER;
BEGIN
END P1;
PROCEDURE P2;
TYPE
Mas=A;
BEGIN
END P2;
```

Здесь описание идентификатора A по тексту предшествует описанию идентификатора Mas, но эти два описания расположены в несвязанных между собой процедурах. Поэтому такое объявление ошибочно.

Использование составных имен

Имя может указывать на единичный объект, как уже было сказано выше и может указывать на составной объект, то есть объект, состоящий из других объектов. Таких сложных конструкций в КП две: записи и объекты. Составной частью записи является поле, составное частью объекта – метод. Доступ к имени, являющимся частью сложного имени, осуществляется через точку.

Имя объемлющей структуры «.» Имя вложенной структуры
Вложенная структура также может быть составной.

Уточнение идентификатора. Объявленный идентификатор может использоваться только в том программном блоке, в котором он был определен. Самый верхний уровень в котором можно выполнить определение, это модуль. Но структура описанная в модуле может быть экспортирована за пределы модуля. Тогда в случае использования ее идентификатора за пределами определяющего модуля, идентификатор должен быть уточнен идентификатором модуля. Если, например идентификатор A экспортирован из модуля M, то за пределами модуля M обращение к нему выполняется так: M.A

Величины. Типы данных. Объявление и виды типов

В перечне основных понятий мы определили программу, как управление вычислением выражений. Выражение содержит операции, константы, константные выражения, скобки. Поэтому прежде чем приступать к разбору проблем управления (операторов языка) необходимо дать детальное представление перечисленных понятий.

Значение выражения это величина некоторого типа. Поэтому, прежде разговора о величинах, необходимо дать представление о типах.

Информация о типе формируется в КП несколькими способами. Во-первых, существуют так называемые основные типы, – это простейшие типы представляющие собой наиболее часто встречающиеся виды данных. Фактически это то, без чего невозможно написание простейшей программы: некоторые виды чисел и литеры. Из основных типов формируется два вида составных: массивы и записи из которых можно составлять сложные структуры данных. КП, также позволяет формировать объявление типа, такое объявление описывает не свойства величин, а свойства типов.

Величины, описываемые непосредственно в блоке объявлений называются статическими величинами. Память под них выделяется на этапе компиляции. Есть в КП возможность объявить динамическую величину, то есть величину описанную, но пока не существующую, до особого распоряжения программиста. Типы динамических величин определяются также, как и типы статических. Динамическая величина, это не особенный тип, это способ объявления и способ существования величины.

И наконец в КП существуют так называемые константы – величины для которых на этапе компиляции определяется не тип, а сразу значение и уже по значению компилятор принимает решение о объеме выделяемой памяти и что с этой величиной можно делать. Начнем изучение величин с констант.

Согласно сообщению о языке, описание константы связывает идентификатор с неизменяемым значением. Неизменяемость значения означает возможность определения значения на этапе компиляции. Блок описания констант начинается ключевым словом CONST.

Примеры правильного определения констант:

```
CONST
   N=5;
   A=12.45;
   C=4*N+6*A;
```

Константа C определена только в том случае, если определены константы используемые в ее описании. Порядок описания существенно важен. Следующее описание

```
CONST
   C=4*N+6*A;
   N=5;
   A=12.45;
```

с точки зрения компилятора ошибочно. При обсуждении свойств идентификаторов утверждалось, что объявления величин могут записываться непоследовательно при условии, что они записаны в одном блоке. В примере на первый взгляд именно такая ситуация, величины A, N, C описаны в одном блоке, но тем не менее это ошибка. Все же ситуация немного отлична. Правило объявления идентификаторов работает действительно именно так, как было указано, но здесь мы имеем дело не с объявлением, а определением. Отличие в том, что при определении указывается и значение величины.

Поэтому здесь работает правило определения константных выражений которое говорит, что константное выражение возможно вычислить простым просмотром (правило дано чуть ниже). А это возможно только в том случае, если к моменту компиляции все величины входящие в выражение имеют точно определенные значения.

Тип константы не указывается, определенность значения константы на этапе компиляции позволяет компилятору выделить всю необходимую информацию из вычисленного значения. Описание константы C показывает также, что значением константы может быть константное выражение. Это допустимо согласно определения константы, так как константное выражение, также имеет неизменяемое значение.

Константные выражения

В сообщении о КП дано следующее определение: *константное выражение* – это выражение, которое может быть вычислено при простом текстуальном просмотре

без фактического выполнения программы. Его операнды суть константы или предопределенные процедуры – функции, которые могут быть вычислены при компиляции.

Формальное определение

```
ConstantDeclaration     = IdentDef "=" ConstExpression.
ConstExpression         = Expression.
```

Примеры константных выражений уже приводились выше, поясним лишь, что предопределенные процедуры – функции это функции известные компилятору, именно поэтому их значение может быть вычислено на этапе компиляции, естественно при условии, что аргумент таких функций, также константное выражение.

Антипод понятия константа – понятие переменной. Из самого термина ясно, что переменная – это величина, чье значение может изменяться в ходе выполнения программы. Поэтому на этапе компиляции определяется тип переменной, и не определяется значение. Объявление переменой согласно сообщения о языке выглядит так:

```
VariableDeclaration = IdentList ":" Type.
```

IdentList – это список имен переменных. Поэтому одно объявление может быть применено к группе имен переменных. Далее за двоеточием следует имя типа.

Имя переменной величины – это идентификатор, поэтому правила формирования имен и возможные ограничения следует смотреть в параграфе описания идентификаторов. A объявление типа согласно сообщению о языке имеет следующий вид:

```
TypeDeclaration     = IdentDef "=" Type.
Type = Qualident | ArrayType | RecordType | PointerType | ProcedureType.
```

TypeDeclaration – объявление типа
IdentDef – определяемый идентификатор
Type – тип
Qualident – уточненный идентификатор
ArrayType – тип массива
RecordType – тип записи
PointerType – Указательный тип
ProcedureType – Процедурный тип

Особенно следует обратить внимание на термин Qualident. Уточнить тип идентификатора возможно, как основным типом, так и собственным. То есть под определение Qualident попадает объявление следующего вида:

```
TYPE
    Mas=ARRAY 10 OF INTEGER;
    MyMass=Mas; (*уточненный идентификатор MyMass*)
```

Следует также обратить внимание на процедурный тип. Величины этого типа связываются не со структурами данных, а процедурами. В этом процедурный тип стоит особо от других типов данных. Необходимо также в отношении процедурного типа заметить, что он в настоящее время считается устаревшим средством и оставлен в КП только для поддержки уже разработанного ПО. Согласно сообщению, поддержка процедурных типов может быть сокращена в следующих версиях языка. Поэтому для программистов выбравших в качестве языка язык семейства Паскаль видимо следует воздерживаться от употребления этого средства при разработке нового ПО.

Описание типов переменных величин, начнем с основных. Как уже было сказано выше, основные типы предназначены для описания различных видов чисел и литер. Все остальные возможности вынесены в составные типы и собственные. Основные типы перечислены в следующей таблице.

Основные типы данных

Таблица 2.2. Список основных типов данных

Имя типа	Значения
BOOLEAN	логические значения TRUE и FALSE
SHORTCHAR	Литеры набора Latin-1 (0X .. 0FFX)
CHAR	Литеры набора Unicode (0X .. 0FFFFX)
BYTE	целые от MIN(BYTE) до MAX(BYTE)
SHORTINT	целые от MIN(SHORTINT) до MAX(SHORTINT)
INTEGER	целые от MIN(INTEGER) до MAX(INTEGER)
LONGINT	целые от MIN(LONGINT) до MAX(LONGINT)
SHORTREAL	Вещественные числа от MIN(SHORTREAL) до MAX(SHORTREAL), значение INF (INF предопределенное значение которым замещается вещественная величина в случае выхода за пределы допустимого интервала. Знак INF совпадает со знаком исходного значения)
REAL	Вещественные числа от MIN(REAL) до MAX(REAL), значение INF
SET	множества целых чисел из диапазона от 0 до MAX(SET)

Функции MIN и MAX используются для описания интервалов типов в силу того, что реальное значение границ интервалов зависит от реализации. Указанные функции используются для определения границ интервалов основных типов.

Основные типы образуют иерархию типов. Смысл иерархии в следующем: если тип A является младшим по отношению к типу B, то величина типа A может быть присвоена величине типа B. Например: Вещественная величина:=Целая величина, но никак не наоборот. Целые типы являются младшими по отношению к действительным. В КП есть две цепочки иерархии для чисел и для литер.

Иерархия числовых типов:

REAL>SHORTREAL>LONGINT>INTEGER>SHORTINT>BYTE

Иерархия литерных типов:

CHAR>SHORTCHAR

Для составных типов понятие иерархии не определено.

Составные типы. Типы массивов

Массив – структура, являющаяся упорядоченным множеством элементов одного и того же типа. Количество элементов массива называется его *длиной*. Обращение к элементам массива выполняется с помощью индексов, являющихся целыми числами из диапазона от 0 до длина – 1. В сообщении о языке дано следующее определение:

```
ArrayType   = ARRAY [Length {"," Length}] OF Type.
Length      = ConstExpression.
```

Описание длины может отсутствовать, так как длина указана в квадратных скобках. Если длина не указывается, то такой массив называется открытым. Длина открытого массива определяется в процессе работы программы, из чего следует, что открытый массив применяется только в следующих ситуациях:

- для объявления указательного типа;
- для объявления типа элемента открытого массива;
- для объявления типа формального параметра в процедуре.

Описание длины представляет собой список констант, следовательно данное описание предполагает возможность многомерных массивов. Количество размерностей правилами языка не ограничено и определяется только объемом доступной памяти.

Тип вида

```
ARRAY L0, L1, ..., Ln OF T
```

интерпретируется как сокращенная запись для

```
ARRAY L0 OF
   ARRAY L1 OF
   ...
      ARRAY Ln OF T
```

Этот вид записи можно воспринимать и как многомерный массив, элементами которого являются элементы указанного типа и как массив массивов, в качестве элементов которого можно использовать массивы меньшей размерности.

Например при следующем описании:

```
mas1, mas2: ARRAY 10,10 OF INTEGER;
```