http://vmk.ucoz.net/

ГЛАВА 4

Программа на языке Object Pascal

> Pascal is very elegant. It's certainly still alive. It is prolific of successors and it has influenced language design profoundly.
> (Паскаль весьма изящен. Конечно, он до сих пор жив. У него тьма последователей, и он существенно повлиял на разработку языков программирования.)
>
> *Dennis M. Ritchie*

Свершилось! Пробившись сквозь дебри анализа предметной области, преодолев изучение классификации средств разработки, ознакомившись с некоторыми особенностями аппаратуры и современных операционных систем, мы, наконец-то, добрались до детального знакомства с языком программирования Object Pascal.

В данной главе мы заложим весьма прочный фундамент здания под названием «Программирование с использованием языка Pascal», а в последующих главах возведем на этом фундаменте несколько этажей. Впрочем, как известно, чем выше окно, тем дальше из него видно, так и в нашем случае – чем больше мы будем узнавать, тем большая перспектива будет перед нами открываться. Надеемся, что к концу книги вы сможете с удовлетворением сказать: «Как много пройдено! Какое счастье, что впереди еще больше!».

1. Историческая справка по языку Pascal

В ноябре 2000 года исполнилось 30 лет с момента первой официальной публикации описания языка Pascal. Произошло это событие в стенах Швейцарского федерального технологического института (Eidgenoessische Technische Hochschule – ETH), а сама публикация представляла собой недоступный широкой аудитории технический отчет. В том же 1970 году был написан первый компилятор языка Pascal (ETH Pascal). А в конце года

Никлаусом Виртом[1], по праву считающимся отцом-основателем языка, было опубликовано первое официальное его описание с изложением синтаксиса и семантики.

В самом начале 1971 года упомянутый выше отчет был перепечатан в первом номере журнала «Acta Informatica» [42]. Так что датой рождения нового языка можно считать и этот момент [1].

Созданный изначально как язык для обучения студентов, позднее Pascal получил признание и в качестве коммерческого средства разработки программ. Так, в период с 1975 по 1980-е годы активно использовалась реализация под названием UCSD Pascal (Кеннет Боулес) [27].

Однако настоящую массовость использованию языка сумела обеспечить компания Borland International. Созданные под руководством профессора Андерса Хейльсберга среды программирования под общим названием Turbo Pascal (последняя версия среды имела номер 7.0) являлись одним из основных средств разработки программ для архитектуры x86 и операционных систем семейства DOS. Borland существенно расширила разработанный Виртом язык, сделав его мощным и удобным инструментом разработки, имевшим развитые средства поддержки технологий структурного и модульного программирования, а также несколько упрощенные возможности программирования в рамках объектно-ориентированного подхода. В последующих главах мы подробно рассмотрим основные положения данных технологий, причины их появления и преимущества, получаемые от их использования на практике.

С появлением первой завоевавшей массовую популярность операционной системы семейства Windows – Windows 3.1 компания Borland адаптировала под нее среду Turbo Pascal 7.0, переименовав ее попутно в Borland Pascal и предоставив разработчикам инструментарий для создания оконных приложений в виде библиотеки OWL – Object Windows Library.

К счастью, и на этом разработчики компании Borland не остановились – в начале 90-х годов язык Pascal подвергся, наверное, самой серьезной переработке с момента своего возникновения и с полным основанием был переименован в Object Pascal, став основой новой среды разработки Borland Delphi. Описание всех нововведений потребовало бы от нас отдельной главы, упомянем лишь весьма значительные изменения в поддержке объектно-ориентированного программирования и, конечно, качественную и

---

[1] Никлаус Вирт (Niklaus K. Wirth) – швейцарский ученый, профессор Швейцарского федерального технологического института, идейный вдохновитель и непосредственный участник процесса эволюции семейства основанных на Pascal языков программирования.

продуманную реализацию новых на тот момент концепций визуального и компонентного программирования[1], что привело, с одной стороны, к огромной популярности среды разработки, а с другой – впервые сделало создание программ под Windows по-настоящему доступным[2].

Сегодня система Borland Delphi уверенно занимает свой сегмент рынка, по-прежнему используя Pascal в качестве базового языка (хотя компания Borland в последнее время называет сам язык Delphi).

Наряду с описанной ветвью развития языка продолжались научные разработки и в области создания новых языков программирования на основе Pascal. Так, хорошо известна разработанная под научным руководством Вирта линейка «Pascal – Modula-2 – Oberon – Oberon-2» [32, 43, 44].

В последние годы начала активно распространяться новая платформа разработки .NET, созданная и продвигаемая на рынок компанией Microsoft. Технология .NET позиционируется как средство создания переносимых распределенных (в том числе Web) приложений и предоставляет в числе прочих возможности разработки многоязыковых программ. В рамках технологии .NET в институте ETH под руководством профессора Дж. Гудкнехта (J. Gutknecht) разрабатывается компилятор нового языка программирования Zonnon, основанного на Pascal и предназначенного для использования в рамках платформы .NET [2, 29–31]. Стоит заметить, что компания Borland также не осталась в стороне. Последняя их среда разработки Borland Delphi 2005 не только полностью интегрирована с .NET, но и содержит два базовых языка: Object Pascal и C#[3] – новый язык, разработанный компанией Microsoft специально под платформу .NET [25].

2. Синтаксическая характеристика языка

Любой язык представляет собой средство общения и передачи информации. В подавляющем большинстве естественных языков существуют две иногда весьма отличающиеся друг от друга формы: устная и письменная. Причем первый язык общения каждый человек изучает в устной

---

[1] Справедливости ради необходимо отметить, что первыми реализацию «визуального» подхода к построению программ осуществила компания Microsoft для языка Visual Basic.

[2] Тот, кто хотя немного знаком с WinAPI – этим краеугольным камнем, на котором зиждется программирование под операционные системы семейства Windows, – нас поймет.

[3] Читается «си шарп».

форме и лишь затем узнает о существовании письменности, формальных правил словообразования, построения предложений, орфографии, пунктуации и т.д.

Языки программирования есть «средство общения человека и компьютера», а потому должны быть максимально формализованы, любая конструкция языка должна иметь четкий однозначно понимаемый смысл. И, конечно же, языки программирования существуют исключительно в письменной форме – по крайней мере, до тех пор, пока компьютер не «научат» понимать обычную человеческую речь.

В отличие от первого языка общения, освоение которого приходится у человека на фактически бессознательный возраст, изучение второго и последующих языков обычно начинается именно с письменного варианта: алфавит, набор базовой лексики, грамматика. Именно в таком порядке и мы будем знакомиться с языком Pascal.

2.1. Методы описания синтаксиса (БНФ, синтаксические диаграммы)

Под *синтаксисом языка* формально понимается набор правил, разъясняющих, какие последовательности из символов алфавита языка являются допустимыми.

Под *алфавитом* понимается набор «атомарных» символов (букв), из которых может быть построен текст на алгоритмическом языке.

Описание синтаксиса языка представляет собой некоторую проблему. Если определить алфавит можно просто перечислив все символы, число которых конечно и, более того, относительно невелико, то указать все допустимые цепочки символов, т.е. перечислить все правильные программы, практически невозможно. Что же делать? Решение – сформулировать *правила построения* допустимых цепочек.

Формальная спецификация абсолютно необходима для любого языка программирования высокого уровня, поскольку программы, написанные на этом языке, должны автоматически переводиться транслятором в машинный код. Для подавляющего большинства популярных языков количество трансляторов несколько больше, чем один, хотя бы потому, что популярные языки используются на разных платформах, то есть под разными операционными системами, а иногда и на разной аппаратной базе. Поскольку одно из основных достоинств языков программирования высокого уровня – независимость или, говоря по-другому, переносимость, необходимо, чтобы любая реализация транслятора совершенно одинаково интерпретировала любую конструкцию языка.

Для строгого описания синтаксических правил изобретены специальные языки, получившие наименование *метаязыки* («надязыки»). Такое

название связано с их функцией «языка для описания языка». Наиболее широко употребляемыми из них являются *металингвистические формулы Бэкуса–Наура* (язык БНФ) и *синтаксические диаграммы*.

Форма Бэкуса–Наура – формальная система описания синтаксиса, в которой одни синтаксические категории последовательно определяются через другие.

Впервые была использована для описания языка программирования Algol-60. Известна также *расширенная форма Бэкуса–Наура*, в которой для удобства изложения добавлено некоторое количество конструкций.

Полное изложение БНФ увело бы нас далеко в сторону, здесь мы ограничимся лишь парой примеров.

Вот как на языке БНФ может быть описано правило, определяющее цифру:

<цифра> ::= 0|1|2|3|4|5|6|7|8|9.

Здесь знаки « < », « > », « ::= » и « | » являются метасимволами языка БНФ и используются в следующем смысле: угловые скобки ограничивают понятия описываемого языка (метапеременные), знак « ::= » читается как «по определению есть» или просто «это», а вертикальная черта читается как «либо» («или»).

А вот так может быть описано правило, определяющее синтаксически допустимую последовательность символов алфавита (в данном случае арабских цифр), для изображения целого числа без знака:

<целое без знака> ::=
<цифра> | <целое без знака><цифра>

Приведенная формула задает рекурсивное определение целого числа без знака, т.е. определение через само себя. Рассмотрим, как «раскручивается» такое определение.

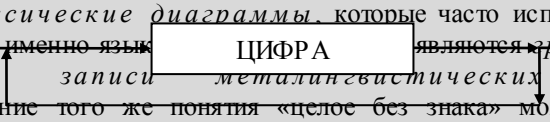
Во-первых, вспомним, что «цифра» уже определена нами ранее.

Во-вторых, заметим, что первым (до знака « | ») стоит определение целого числа без знака как цифры. Тогда становится понятно, что вторая часть определения поясняет, что для построения правильного целого числа следует справа к числу (а первый раз это одна цифра) приписывать какую-нибудь цифру.

Очевидным недостатком языка БНФ с точки зрения обучения является невозможность, например в нашем случае с целым числом, указать,

сколько же может быть цифр в числе. С некоторой точки зрения это неважно (ниже мы поясним, о чем идет речь), но человека, изучающего язык по металингвистическим формулам, такой способ определения целого числа может ввести в заблуждение.

Синтаксические диаграммы, которые часто используются для описания именно языков, являются *графической формой записи металингвистических формул*. Определение того же понятия «целое без знака» может выглядеть следующим образом:



где метапеременная ЦИФРА заключена в прямоугольник, а ее повторяемость графически изображена обратной стрелкой, задающей цикл повторений.

Не останавливаясь на обсуждении достоинств и недостатков приведенных метаязыков, укажем главное. Основное их назначение не связано с изложением языка для пользователя. Они решают другую задачу – обеспечивают возможность формального описания элементов и конструкций языка для их однозначной интерпретации. При этом понятно, что разработчику транслятора не надо, например, пояснять, сколько может быть цифр в целом числе. Он реализует представление целых чисел в соответствии с принятыми спецификациями.

Для пользователя же языка более приемлемым является словесное неформальное описание языковых конструкций, сочетающее синтаксис и семантику, т.е. правила истолкования, смысл предложений. Именно таким способом мы и будем пользоваться в дальнейшем, иногда прибегая к угловым скобкам для обозначения метапеременных, а также к еще одним метасимволам – квадратным скобкам, обозначающим, что заключенная в них конструкция может быть опущена. Заметим, что и угловые скобки (символы «меньше» и «больше»), и квадратные скобки входят в алфавит языка Pascal.

2.2. Алфавит языка

Чтобы получить общее представление об *алфавите языка*, достаточно взглянуть на клавиатуру компьютера – большинство из изображенных там символов входит в «азбуку» любого языка программирования. Конечно же, и Pascal не является исключением из этого правила.

Мы не будем перечислять здесь все допустимые символы, а сделаем лишь несколько общих замечаний.

Во-первых, обратим внимание на использование русских букв. Их допускается применять только в комментариях к программе и в строковых и символьных константах. Никакие другие языковые конструкции не могут содержать русские буквы.

Во-вторых, отметим, что за исключением символьных и строковых значений компилятор в полном соответствии с правилами языка не различает прописные и строчные буквы, то есть слова **Program** и **program** в языке Pascal являются идентичными. Таким образом, использование букв в разных регистрах является лишь вопросом стиля записи программы и дополнительной возможностью повышения ее «читабельности», о чем мы будем далее неоднократно упоминать.

2.3. Спецсимволы

Спецсимволы – символы и пары символов, которые имеют специальное, заранее определенное значение.

К числу спецсимволов языка Pascal относятся следующие символы и пары символов:

```
# $ & ' ( ) * + , - . / : ; < = > @ [ ] ^ { }
(* (. *) .) .. // := <= >= <>
```

Следующие символы и пары символов являются эквивалентными:

```
[     (.
]     .)
{     (*
]     *)
```

2.4. Ключевые слова

В известном смысле в алфавит языка входит и ряд так называемых *ключевых (зарезервированных, служебных) слов* – в примерах программ, которые рассматриваются на протяжении данной книги, они выделены жирным шрифтом. Эти слова не могут быть использованы ни в каком другом смысле, кроме изначально приписанного разработчиками языка. Пояснять значение этих слов мы будем по мере необходимости. Заметим, что выделение нами ключевых слов с помощью

жирного шрифта служит лишь иллюстративным целям. В реальной практике характер выделения ключевых слов (или отсутствие такового) целиком зависит от текстового редактора, который используется для работы с кодом программы.

```
and                 array               as
asm                 at                  automated
begin               case                class
const               constructor         destructor
dispinterface       div                 do
downto              else                end
except              exports             file
finalization        finally             for
function            goto                if
implementation      in                  inherited
initialization      inline              interface
is                  label               library
mod                 nil                 not
object              of                  or
on                  out                 packed
procedure           program             property
private             protected           public
published           resourcestring      set
raise               record              repeat
shl                 shr                 string
then                threadvar           to
try                 type                unit
until               uses                var
while               with                xor
```

2.5. Идентификаторы

Любой объект программы (переменная, константа, тип данных, процедура и т.д.) должен иметь имя. В языках программирования имена называются *идентификаторами*.

Правила записи идентификаторов различаются не только от языка к языку, но и в разных версиях одного и того же языка. На данный момент в языке Object Pascal эти правила состоят в следующем:

- идентификатор может начинаться только с буквы или символа подчеркивания;
- далее могут следовать буквы, цифры и знак подчеркивания;

- формально длина, т.е. количество символов, идентификатора не ограничена, однако только первые 255 являются значащими. Другими словами, два разных идентификатора должны иметь различия в первых 255 символах, что, конечно, не является серьезным ограничением с практической точки зрения.

Сразу отметим, что хороший стиль программирования подразумевает выбор имен не «с потолка», а в соответствии со смыслом обозначаемого объекта. При этом рекомендуется пользоваться прописными буквами и символом разделения – подчеркиванием. Не следует также бояться (а точнее говоря, лениться) записывать длинные имена – это окупится большей ясностью программы.

Приведем примеры идентификаторов:

```
ListOfPersons, Window_12, cpu86.
```

Еще раз обратим внимание, что в именах нельзя использовать русские буквы, а также на то, что имена не должны совпадать с ключевыми словами.

Заметим следующий факт: многие начинающие программисты (конечно, мы далеки от мысли, что вы считаете себя начинающим ☺. В этом случае вам проще будет осознать то, что сейчас будет написано…) не придают никакого значения проблеме правильного именования.

Удивительно, как много программ содержат имена вроде `aaa`, `qqq`, `qweqwe`, `qq11`, `a1`, `a2`, `a22`, `Button1`, `Form1` и т.д. Неизбежным следствием этого безобразия является превращение текста программы в некую шифрограмму, что нивелирует сам смысл применения языка программирования высокого уровня. Мы берем хороший инструмент и пренебрегаем его возможностями. Что можно понять, глядя на следующую строку текста программы?

```
qqq := (a1 + a2) / 2 * StrToFloat(Edit.Text);
```

С определенностью лишь одно – здесь что-то складывается, делится и умножается, а вот что именно, зачем и почему, можно выяснить только путем «долгого и мучительного» анализа контекста для выяснения смысла, заложенного автором в каждую из участвующих в выражении переменных. Непонимание или хуже того – игнорирование этого факта, конечно же, выйдет боком самому горе-программисту, когда он через месяц попытается модифицировать свою программу, добавляя в нее новую функциональность или устраняя внезапно обнаруженные ошибки.

Хотелось бы сразу избавить читателя от иллюзий: мысль о том, что «мы сейчас сделаем как-нибудь, а вот когда понадобится, будем делать по-человечески» на практике нереализуема. Когда доходит до дела, люди, привыкшие работать по принципу «тяп-ляп», с большим трудом

перестраиваются. Таким образом, с нашей точки зрения, совершенно необходимо сразу привыкнуть присваивать всем объектам содержательные имена – в дальнейшем эти усилия окупятся сторицей.

2.6. Объявления

Объявления – специальные выражения языка, указывающие на то, что мы хотим далее в своей программе использовать идентификатор в тех или иных целях. Сам идентификатор и цели использования вытекают из содержимого объявления.

```
var
   Number: Integer;
const
   MinNum = 10;
   MaxNum = 1000;
type
   TElement = Word;
   TMyArray = array[MinNum..MaxNum] of TElement;
```

Общая идея, реализованная в языке Pascal повсюду (существующие исключения мы рассмотрим по мере необходимости), заключается в том, что мы должны «сначала объявить – затем использовать». Так, обязательны предварительные объявления переменных, констант, новых типов данных, подпрограмм и др.

2.7. Операторы

Понятие оператора является, с одной стороны, основным в большинстве языков программирования, с другой – дать ему четкое определение весьма непросто. Попробуем дать определение конструктивное. Прежде всего, *оператор* – любая *исполняемая* конструкция языка программирования. Некоторые такие конструкции являются составными, то есть включают в себя другие операторы. Довольно часто набор из нескольких операторов, синтаксически или семантически связанных, также называется оператором. Как видим, определение получается рекурсивным. Подводя итог, можно сказать, что оператор есть выраженное средствами языка программирования действие.

Естественным образом возникает вопрос: «Сколько операторов должно быть в языке?». Разговор о том, сколько операторов *достаточно*, мы

отложим до следующей главы, а сейчас подумаем над таким возможным ответом: «Чем больше, тем лучше». Казалось бы, ответ разумен, чем больше операторов, тем более выразительным становится язык программирования. Однако здесь скрывается существенная проблема.

Представьте себе, что вы вместе с коллективом программистов работаете над большим проектом. Представьте, что средством разработки служит воображаемый язык «SuperPascal nonsense edition», в котором определена пара сотен операторов. Будете ли вы знать их все? Честно говоря, на этот счет нас «терзают смутные сомнения». Скорее всего, дело ограничится десятком-другим операторов, тех, что наиболее часто используются, а за остальными придется при необходимости обращаться к справочникам. В результате в коллективе неизбежно возникновение несовпадающих подмножеств операторов, известных разным людям, что существенно затруднит совместную работу над проектом.

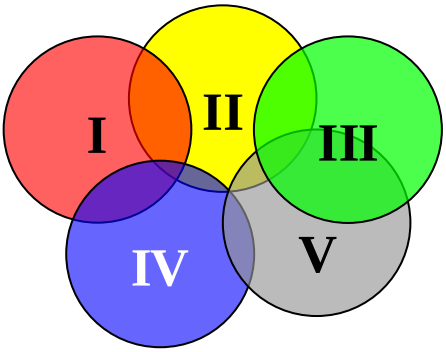


Рис. 4.1. Перекрытие областей знаний программистов для операторов языка «SuperPascal nonsense edition»

Отсюда вывод – да здравствует разумная достаточность!

Откроем маленький секрет – этого правила обычно придерживаются и создатели языков программирования, большая их часть имеет сравнительно небольшой набор операторов. В следующей главе мы узнаем, чем этот набор определяется.

2.8. Комментарии

Комментарии – это пояснения к программе. Комментарии не анализируются компилятором и никак не влияют на ее исполнение. Комментарий может быть записан в любом месте программы, где может

стоять пробел. Правила его записи следующие: комментарий заключается либо в фигурные скобки: { и }, либо в ограничители, состоящие из круглых скобок со звездочками: (* и *), внутри скобок могут находиться любые символы, включая и русские буквы.

```
(*  Все это комментирующий текст *)

{   И это тоже }

(*
    И даже это
*)

{
    И, наконец, это
}
```

С появлением систем визуального программирования Delphi стало допустимым использование в тексте программы на языке Object Pascal комментариев в стиле C++, в частности текст в строке, следующий за // , является комментарием.

```
// Комментарий
```

Тот, кто изучал программирование, наверное, уже не один раз слышал и читал настоятельную рекомендацию: «Пишите комментарии!». Причина этого призыва понятна. Программа – это не только инструкция для работы компьютера, но и документ, фиксирующий идеи решения задачи, часто весьма нетривиальные. Программа может быть и часто бывает использована как способ сообщения знаний другому лицу. Кроме того, и сам программист через некоторое время забывает то, как он реализовывал те или иные моменты и, если приходится возвращаться к старой некомментированной программе, испытывает определенные затруднения в ее понимании.

С другой стороны, авторы знают и на собственном опыте, и из бесед с другими программистами, имеющими большой стаж работы, как, несмотря на понимание необходимости использования комментариев и даже несмотря на свой печальный опыт их игнорирования, тем не менее, трудно заставить себя отвлечься на некоторую вспомогательную «черновую» работу, будучи увлеченным составлением хитроумного алгоритма и его оперативной отладкой. Здесь можно дать только один совет. Вставляйте комментарии не сразу, а после того как записан и отлажен некоторый фрагмент программы. Но делайте это обязательно!

Наконец, последнее замечание. Комментарии должны дополнять текст программы, а не разъяснять правила работы операторов. Например, комментарий типа

```
{ складываются числа Segment и Offset }
```

следует признать крайне неудачным, так как предполагается, что читающий программу и так знает язык и, соответственно, может понять, как работает оператор, особенно если за идентификаторами закреплен известный читателю смысл.

Если говорить о некоторой общепринятой практике, то обычно в первую очередь записываются комментарии к программе в целом, к отдельным подпрограммам, а также дается содержательное описание объектов программы.

2.9. Директивы компилятору

Язык Pascal содержит специальную синтаксическую конструкцию, позволяющую управлять компиляцией программы, точнее, выдавать команды компилятору по включению в исполняемый модуль или исключению из него тех или иных групп вспомогательных машинных команд. Пусть, например, программа должна обрабатывать некоторую информацию, хранящуюся на жестком диске в виде файла. Если имя этого файла должен задать пользователь программы, он вполне может ошибиться при его вводе. В этом случае при отсутствии специальных указаний компилятор автоматически вставит в исполняемый модуль команды, анализирующие характер завершения операции поиска файла и вызывающие принудительное завершение программы, если файл не был найден. Очевидно, что в подавляющем большинстве случаев ситуация неверного ввода имени файла не является критической и вместо аварийного выхода из программы пользователю нужно предложить ввести имя файла заново. Тогда нам и понадобятся директивы, чтобы «объяснить» компилятору, что мы не нуждаемся в автоматической обработке, а будем выполнять ее сами.

Синтаксически директивы в языке Pascal записываются в фигурных скобках, подобно комментариям, однако сразу же после открывающей скобки должен стоять символ «знак доллара», т.е. `$`. В этом случае компилятор не игнорирует данный текст, а рассматривает его как указание к действию. После знака `$` следует имя директивы – одна буква или развернутое наименование (некоторые директивы имеют оба варианта) – и признак включения или выключения действия: знаки плюс, минус или слова `ON`, `OFF`. В дополнение к этому можно записать произвольный текст,

скажем для пояснения директивы. В случае неверного написания имени директивы компилятор выдаст сообщение об ошибке. Конкретные директивы мы будем рассматривать по мере необходимости. А сейчас приведем пример директивы отключения контроля выполнения операций ввода-вывода, использованной, кстати, еще в примере 1.1:

```
{$I – отключить автоматический контроль ввода-вывода}
```

Директива может также выглядеть следующим образом:

```
{$WARNINGS OFF}
```

2.10. Структура программы

Синтаксис языка Object Pascal значительно более «либерален», чем тот, что был определен изначальным стандартом языка Pascal. Программа в Object Pascal состоит из операторов или предложений языка, которые разделяются между собой точкой с запятой « ; » и записываются в свободном формате. Компилятор рассматривает программу как последовательность строк, каждая из которых содержит последовательность символов и включает ряд операторов, между которыми может находиться сколько угодно пробелов. Пользуясь возможностями свободного формата, программист произвольным образом разбивает текст на строки, руководствуясь размерами экрана (или размерами листа бумаги, на котором будет печататься текст) и соображениями хорошего стиля записи программ. Для большей наглядности программы в нее вставляются пустые строки, отделяющие фрагменты алгоритма, используются отступы при записи операторов (все примеры в книге выдержаны в некотором общем стиле, включающем, помимо прочего, и данное правило).

В общем случае последовательность записи элементов программы должна выглядеть следующим образом (впоследствии эта схема будет уточняться):

```
[Program <Имя программы>;]
[uses <список имен используемых библиотек>;]
   [<объявления меток>]
   [<объявления констант>]
   [<объявления типов>]
   [<объявления переменных>]
   [<объявления и описание процедур и функций>]
begin
   [<исполняемые операторы>]
end.
```

Программа начинается (если не считать заголовочных комментариев) с ключевого слова **Program**, за которым следует имя программы –

идентификатор. В языке Object Pascal это предложение можно опустить, о чем говорят метасимволы – квадратные скобки. Надо сказать, что имя программы, задаваемое в заголовке, с точки зрения ее выполнения не играет никакой роли (поэтому его и можно опускать). Тем не менее, мы настоятельно рекомендуем начинать текст программы с оператора **Program** и использовать его для указания в названии программы ее назначения.

После заголовка программы следует необязательная секция подключения библиотек, начинающаяся с ключевого слова **uses** («использует»). Более подробно мы поговорим об этом в главе 7. На практике только очень простые программы могут обойтись без использования этой секции.

Далее по порядку следуют разделы различных объявлений. Еще раз отметим, в языке Pascal *все* объекты в обязательном порядке должны быть *объявлены*. Ни один объект не может быть использован ни в каком качестве без предварительного объявления. Каждый из указанных выше разделов объявлений мы рассмотрим достаточно подробно. Пока же отметим, что в отличие от стандарта языка в версии Object Pascal разделы объявлений могут располагаться в произвольном порядке и даже вперемешку, т.е. например, между разделами объявления переменных может находиться раздел с определениями типов. Единственное правило должно соблюдаться неукоснительно: «сначала объяви, потом используй» (правда, как было сказано ранее, и здесь есть исключения).

Наконец, самым последним (это правило нарушено быть не может) следует раздел операторов – *тело программы*. Он начинается с ключевого слова **begin** и завершается ключевым словом **end** с последующей точкой, которая собственно и является признаком конца текста программы, так как слово **end** может встретиться и в другом контексте.

Из приведенной схемы нетрудно заключить, что минимальной программой на языке Pascal (ничего не делающей, но и не содержащей ни одной ошибки) является программа:

```
begin
end.
```

В заключение отметим, что совокупность разделов описаний и операторов (без предложения **Program** и заключительной точки) называется *блоком*.

3. Типы данных

3.1. Информация и формы представления данных

Цель решения любой задачи – получение новой *информации*. Информация заключается в данных и извлекается из них путем их анализа. Другими словами, данные и информация, вообще говоря, не одно и то же. Информация дополнительно подразумевает существование некоторого способа интерпретации данных, вследствие чего возникает необходимость в наиболее информативном их представлении, т.е. представлении в виде, максимально облегчающем интерпретацию.

Минимальной единицей измерения информации является *бит* (сокращение от *binary digit*).

Один бит соответствует двум различным значениям, обычно обозначаемым 0 и 1. Восемь бит составляют 1 *байт*, 2^{10}=1024 байт – 1 *килобайт*, 2^{10}=1024 килобайт – 1 *мегабайт*, 2^{10}=1024 мегабайт – 1 *гигабайт* и т.д.

3.2. Понятие типа данных

Как известно, данные в памяти компьютера должны быть представлены в виде последовательностей двоичных чисел, другими словами – в алфавите {0, 1}. Очень немногие из нас способны связно мыслить в таком формате. Во всяком случае, тексты, графики, диаграммы, изображения, звуковые сигналы намного более привычны для нас. Таким образом, говоря о данных, мы должны различать два способа их представления: внутреннее – в памяти компьютера и внешнее – на мониторе, принтере, через звуковую карту и динамики. При этом естественно должны быть средства, осуществляющие перевод данных из одной формы представления в другую. Например, для текстовой или числовой информации такой перевод – задача операторов ввода/вывода.

Остановимся на способах внутренней кодировки более подробно.

Прежде всего, отметим: данные, представляющие различные виды информации, конечно же, и кодируются по-разному, хотя и в одном и том же алфавите.

Классическим примером является различие в подходах к кодировке целых и вещественных чисел, о чем мы частично говорили в предыдущей главе. Казалось бы, поскольку множество целых чисел есть подмножество чисел вещественных, было бы разумно все расчеты вести исключительно в них. На самом деле это, конечно же, не так. Во-первых, аппаратная реализация команд для работы с целыми числами существенно более проста и эффективна с точки зрения быстродействия, чем для

вещественных чисел. Во-вторых, вещественные числа в их строгом математическом понимании реализовать на сугубо дискретном устройстве, каковым является процессор, просто невозможно, в связи с чем организация корректных вещественных вычислений на компьютерах представляет собой отдельную весьма непростую задачу.

Проиллюстрируем эту проблему на простом примере. Как известно, количество бит (физических носителей двоичной информации в оперативной памяти) для представления числа всегда ограничено аппаратными возможностями машины. Можно провести аналогию с «окошком» калькулятора, в котором можно высветить только определенное количество цифр. Если мы наберем на калькуляторе число 1, а затем разделим на 3, то в окошке появится результат 0.3333333 (если калькулятор рассчитан на представление 8 десятичных цифр). Далее при умножении этого числа на 3 мы получим 0.9999999, что, строго говоря, не равно 1. Итак, (1/3)*3 не равно 1. Нечто подобное происходит и при выполнении операций над двоичным кодом.

Итак, целые и вещественные числа представляются в различных внутренних форматах и имеют разные наборы машинных команд для выполнения арифметических операций.

Обобщим сказанное. Обработка данных выполняется с использованием оперативной памяти. Данные должны быть некоторым образом закодированы в двоичный вид, что автоматически подразумевает наличие различных способов интерпретации одной и той же области памяти в зависимости от вида размещенных в ней данных. Объем памяти, который может быть выделен под хранение любого данного, естественным образом конечен. Наконец, вид данных автоматически определяет допустимые операции их обработки.

Все сказанное и составляет понятие *тип данных*:

1) множество значений;
2) набор операций, применимых к значениям данного типа;
3) способ представления и интерпретации данных в памяти компьютера;
4) размер оперативной памяти, необходимый для хранения данных в памяти компьютера.

Заметим, что не все указанные пункты независимы (очевидно, что пункты 1 и 4 взаимосвязаны). Однако представляется важным указать их все. Далее с каждым пунктом мы познакомимся подробнее.

3.3. Представление чисел. Системы счисления

В данном разделе мы рассмотрим важный теоретический материал, который поможет в дальнейшем разрешать вопросы, связанные с представлением чисел в оперативной памяти, со способами интерпретации располагаемых в ней данных, с вычислением объема памяти, необходимого для хранения данных.

3.3.1. Понятие системы счисления. Со школьных времен многим из нас известно волшебное словосочетание *система счисления*. Правильное понимание этого понятия имеет в нашем контексте весьма существенное значение. А потому попробуем приоткрыть завесу тайны, скрывающую под собой его смысл.

Давным-давно люди поняли, что для адекватного осуществления разных бытовых операций (обмен, учет, торговля и т.д.) необходимы какие-то абстрактные единицы измерения. Так появились *числа*. Трудно сказать однозначно, кто и когда первым ввел это понятие. Однако достоверно известно, что Евклид в своих «Началах» уделил вопросу о числе большое внимание. Большая советская энциклопедия определяет число так: «Число – важнейшее математическое понятие. Возникнув в простейшем виде еще в первобытном обществе, понятие числа изменялось на протяжении веков, постепенно обогащаясь содержанием по мере расширения сферы человеческой деятельности и связанного с ним расширения круга вопросов, требовавшего количественного описания и исследования. На первых ступенях развития понятие числа определялось потребностями счета и измерения, возникавшими в непосредственной практической деятельности человека. Затем число становится основным понятием математики, и дальнейшее развитие понятия числа определяется потребностями этой науки».

С появлением чисел возникла и новая как для математики, так и для лингвистики проблема – как записывать числа? Для обозначения чисел понадобилось придумать графические изображения элементарных количеств. Эти графические обозначения в настоящий момент мы с вами называем *цифрами*. Но наличия одних лишь цифр, конечно же, недостаточно. Необходимы еще правила построения (записи) из них «больших» чисел. Именно эти правила – набор соглашений, принятых людьми, относительно графического изображения числовых величин и способа их интерпретации – и составляют в совокупности *систему счисления*.

Все развитые цивилизации обладали своими системами счисления; историки и археологи обнаружили такие системы в Древнем Египте, Греции, Вавилоне, Риме, Иудее, Индии, у славян, индейцев майя и т.д. К слову, вавилонская система используется всеми нами до сих пор. Да, да, не

удивляйтесь! Именно из Вавилона к нам пришли 60 минут в часе, 60 секунд в минуте, 12 месяцев в году, примерно 30 дней в каждом месяце.

3.3.2. Непозиционные и позиционные системы счисления. Анализируя разные системы счисления, использовавшиеся в древности или применяемые в наши дни, можно обнаружить два вида таковых: *позиционные* и *непозиционные*.

В непозиционных системах положение цифры не оказывает влияния на ту величину, которую она обозначает. Классическим примером непозиционной системы является римская система счисления. Проиллюстрируем определение на примерах:

```
XI = 10 + 1 = 11
IX = 10 – 1 = 9
XXV = 10 + 10 + 5 = 25
```

Из примеров видно, что знак `X` в любом случае обозначает 10, вне зависимости от того, на каком месте он располагается.

Главным недостатком непозиционных систем является сложность и неудобство осуществления арифметических операций.

В позиционных системах, одним из существенных элементов которых является понятие «разряд», ситуация принципиально упрощается, поскольку появляется возможность задать четкие однозначные алгоритмы выполнения операций – известные нам с детства сложение, вычитание и умножение «столбиком», деление «уголком».

3.3.3. Математические основы систем счисления. Рассмотрев содержательную сторону дела, перейдем теперь к формальному математическому описанию проблемы.

Пусть $p \in N$, $p \geq 2$ – *основание* (количество цифр) системы счисления[1].

Пусть $A_p = \{c_0, c_1, \ldots, c_{p-1}\}$ – *алфавит* системы счисления.

Будем называть такую систему счисления *p-ичной системой счисления* с алфавитом A_p, а $c_0, c_1, \ldots c_{p-1}$ – *цифрами* системы счисления.

В качестве графических обозначений цифр обычно используют символы 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 и, если их не хватает, прописные буквы латинского алфавита (если вам и этого не хватит (!), напишите нам, мы поможем вам обойтись в задаче системой счисления с меньшим основанием).

В соответствии с этой трактовкой рассмотрим примеры наиболее употребительных систем счисления:

---

[1] Здесь и далее рассматриваются только позиционные системы счисления.

- $p = 2$, $A_2 = \{0, 1\}$ – *двоичная* система счисления;
- $p = 8$, $A_8 = \{0, 1, 2, 3, 4, 5, 6, 7\}$ – *восьмеричная* система счисления;
- $p = 10$, $A_{10} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ – *десятичная* система счисления;
- $p = 16$, $A_{16} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$ – *шестнадцатеричная* система счисления.

Десятичная система счисления знакома каждому из нас с детства, именно в ней подавляющее большинство человечества выполняет расчеты. Двоичная система знакома «с детства» почти каждому вычислительному устройству, т.к. именно в этой системе представляются данные в памяти компьютера. Отметим простой факт (тем, кому он неизвестен со школы, станет ясно, в чем дело, через пару страниц): чем больше основание системы, тем короче запись числа. Так, шестнадцатеричная система очень удобна для записи машинных адресов.

Важным для математики вопросом является вопрос о связи значения числа с его изображением в виде системы цифр. Проиллюстрируем решение этого вопроса для натуральных чисел.

Известна теорема о том, что любое десятичное натуральное число N можно единственным образом разложить по степеням p так, что

$$N_{10} = a_k p^k + a_{k-1} p^{k-1} + \ldots + a_1 p + a_0, \qquad (4.1)$$

где $a_k \neq 0$; $0 \le a_i \le p-1$, $0 \le i \le k$ и все a_k и p – десятичные числа.

Заменим теперь каждое десятичное число a_i (любое из них меньше, чем p) соответствующей ему p-ичной цифрой c_i и выпишем все цифры слева направо по убыванию степеней числа p. Полученная в результате запись по определению есть *позиционная запись числа N в p-ичной системе счисления*:

$$N_{10} = (c_k c_{k-1} \ldots c_1 c_0)_p. \qquad (4.2)$$

Во избежание неоднозначности индекс при числе есть основание системы счисления, в которой записано число.

3.3.4. Перевод чисел из десятичной системы счисления в другую и наоборот. Из формул (4.1), (4.2) следуют алгоритмы преобразования между десятичной и p-ичной системами счисления. Рассмотрим их подробнее.

1. Для преобразования числа N из системы с основанием 10 в систему с произвольным основанием p необходимо получить разложение (4.1) и заменить все коэффициенты на цифры p-ичной системы счисления. Для вычисления коэффициентов a_i рекомендуется делить число N на p

«уголком», затем частное от деления снова делить на p, и так до тех пор, пока очередное частное не окажется меньше, чем p. При этом необходимо запоминать остатки от деления. После этого справедливо следующее: последнее частное равно a_0, последний остаток – a_1, …, первый остаток равен a_k. После этого необходимо осуществить замену коэффициентов на p-ичные цифры.

2. Для преобразования числа N из системы с произвольным основанием p в систему с основанием 10 необходимо записать разложение (4.1), предварительно заменив все цифры p-ичной системы счисления на их десятичные эквиваленты и произвести операции возведения в степень, сложения и умножения, вычислив результат.

3.3.5. Перевод чисел из системы счисления с основанием *p* в систему счисления с основанием *q*. *В общем случае* перевод чисел между двумя произвольными системами счисления выполняется через промежуточные операции перевода в десятичную систему, что на практике для больших чисел может оказаться довольно проблематичным (по крайней мере, при решении «на бумаге»). Известны некоторые частные случаи, а именно: перевод чисел между двоичной, восьмеричной и шестнадцатеричной системами счисления.

Рассмотрим ***перевод чисел между двоичной и восьмеричной системами***. Используемый для этого прием называется *метод триад* (троек). Суть его заключается в том, что каждой восьмеричной цифре ставится в соответствие комбинация нулей и единиц по следующей таблице.

Таблица 4.1

Метод триад

Восьмеричная цифра	Двоичное представление
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Так, число 127_8 = 001 010 111_2, где ведущие нули, разумеется, необходимо отбросить. При переводе из двоичной системы в восьмеричную необходимо дополнить число слева нулями так, чтобы все его двоичные цифры можно было разбить на группы по три, после чего произвести замену в соответствии с таблицей 4.1.

Рассмотрим теперь ***перевод чисел между двоичной и шестнадцатеричной системами***. Для этого предназначен *метод тетрад* (четверок). Суть его заключается в том, что каждой шестнадцатеричной цифре ставится в соответствие комбинация нулей и единиц по следующей таблице.

Таблица 4.2

Метод тетрад

Шестнадцатеричная цифра	**Двоичное представление**
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Так, число $12A7_{16}$ = 0001 0010 1010 0111_2, где ведущие нули, разумеется, необходимо отбросить. При переводе из двоичной системы в шестнадцатеричную необходимо дополнить число слева нулями так, чтобы все его двоичные цифры можно было разбить на группы по четыре, после чего произвести замену в соответствии с таблицей.

При переводе ***между восьмеричной и шестнадцатеричной системами*** представляется разумным выполнять эту операцию не через десятичную, а через двоичную систему счисления, используя рассмотренные выше методы.

3.4. Классификация типов данных в Object Pascal

Изучив основы представления чисел, перейдем теперь к рассмотрению типов данных, включенных в язык Object Pascal, но прежде сделаем одно замечание общего характера.

На аппаратном уровне компьютер поддерживает существенно ограниченный набор типов данных. В то же время программист, составляя программу на языке высокого уровня, использует более широкий и содержательно интерпретируемый набор типов и записывает операции над ними в символическом виде, а не в машинных командах. Заботу о реализации этих типов на основе имеющихся примитивов берут на себя разработчики компилятора. Такие новые типы иногда называют *абстрактными*, тем самым подчеркивается, во-первых, отсутствие их на аппаратном уровне, во-вторых, тот факт, что программист может отвлечься (абстрагироваться) от их происхождения и пользоваться ими не задумываясь о способе их реализации (разумеется, настоящий программист должен знать, как все устроено изнутри).

Множество типов данных, конечно, не ограничивается типами, обеспечиваемыми языком программирования. Скажем, в задаче обработки изображений в качестве типа данных неплохо было бы иметь тип допустимых «картинок» с набором соответствующих операций. Современные языки программирования, и в их числе Object Pascal, имеют синтаксические средства создания и оформления абстрактных типов данных, которые мы рассмотрим в главе 6.

В целом в редакции языка Object Pascal, являющейся основой среды визуального программирования Delphi 7, на которую ориентируется данная книга, принята следующая классификация типов данных:

- простые: целые, вещественные, символьный, логический, перечислимый, тип-диапазон;
- строковые;
- указатели;
- процедурный тип;
- специальный тип `Variant`;
- структурированные – типы, создаваемые программистом с помощью средств конструирования новых типов данных: множества, массивы, записи, файлы, классы, ссылки на класс, интерфейсы.

Далее мы изучим некоторые стандартные (встроенные) типы языка.

3.5. Встроенные типы данных

Каждый встроенный тип данных характеризуется приписанным ему разработчиками языка именем (идентификатором), набором допустимых значений и синтаксической формой представления констант, а также набором операций над значениями.

Отметим, что три из них – целый, вещественный и строковый – обладают следующей особенностью: каждый тип фактически представляет собой некоторый набор концептуально близких типов, различающихся между собой множеством допустимых значений.

3.5.1. Работа с целыми числами. Рассмотрим типы данных, используемые в Object Pascal для представления целых чисел, а также их параметры.

В соответствии с документацией разработчика компилятора фирмы Borland, наилучшую производительность (скорость вычислений) при обработке целых значений обеспечивают типы данных `Integer` и `Cardinal`. Кроме того, реализована поддержка типов данных `ShortInt`, `SmallInt`, `LongInt`, `Int64`, `Byte`, `Word`, `LongWord`.

Рассмотрим следующую таблицу, характеризующую типы данных.

Название типа данных	Множество значений	Размер
Типы данных, обеспечивающие наилучшую производительность		
`Integer`	-2147483648..2147483647	32 бита
`Cardinal`	0..4294967295	
Дополнительные типы данных		
`ShortInt`	-128..127	8 бит
`SmallInt`	-32768..32767	16 бит
`LongInt`	-2147483648..2147483647	32 бита
`Int64`	$-2^{63}..2^{63}-1$	64 бита
`Byte`	0..255	8 бит
`Word`	0..65535	16 бит
`LongWord`	0..4294967295	32 бита

Внимательное изучение таблицы позволяет сделать несколько выводов.

Первый. Имеются «типы-синонимы»: `Integer – LongInt`, `Cardinal – LongWord`. Вызвана такая ситуация соображениями обратной

совместимости (для поддержки программ, написанных на основе предыдущих версий языка).

Второй. Для размеров памяти в 1, 2 и 4 байта существуют знаковый и беззнаковый варианты типов. Надеемся, вам уже достаточно очевиден тот факт, что конкретное значение числа определяется не только нулями и единицами, из которых оно состоит, но и способом интерпретации этой последовательности двоичных цифр.

Такое многообразие типов прежде всего связано со стремлением предоставить программисту возможность выбора наиболее подходящего под конкретную ситуацию диапазона. В самом деле, если из существа задачи вытекает, что обрабатываемые значения являются неотрицательными и не превосходят 255 (например, возраст человека в годах), то для их представления в целях экономии памяти разумно выбрать тип `Byte`. В то же время не стоит забывать, что числа типов `Integer` и `Cardinal` обрабатываются процессором наиболее эффективно. Есть еще несколько технических тонкостей, которые могут повлиять на выбор типа данных и эффективность программы.

Конечно, саму эффективность можно понимать по-разному – эффективность по быстродействию или эффективность по затратам памяти. Более того, для специалиста по аппаратуре не является секретом тот факт, что объем используемой памяти может существенно повлиять на скорость программы. Вопросы оптимизации программ выходят за рамки данной книги[1]. Здесь мы лишь посоветуем следовать рекомендации разработчиков из фирмы Borland – везде, где представляется возможным, использовать типы `Integer` и `Cardinal`.

Перейдем теперь к вопросам практического использования рассматриваемых типов данных.

Константы целого типа записываются в языке Pascal естественным, принятым в математике, способом. Константа может иметь знак «+» или «–» или не иметь знака. В последнем случае она считается положительной.

В языке Pascal имеются следующие *арифметические операции* над целыми числами:

Наименование	Обозначение
Сложение	+
Вычитание	–

[1] Желающие могут обратиться к прекрасной книге Криса Касперски «Техника оптимизация программ. Эффективное использование памяти» [6], а также к документации по Intel-процессорам http://developer.intel.com.

Умножение	`*`
Деление (целая часть)	`div`
Остаток от деления	`mod`

Приведем примеры выполнения двух последних операций:

- `7 div 3` равно `2`,
- `1 div 2` равно `0`,
- `5 mod 3` равно `2`,
- `-7 mod 3` равно `-1`.

В языке Pascal имеется возможность применять к целым числам так называемые *битовые операции*, которые рассматривают число как последовательность двоичных разрядов.

Наименование	**Обозначение**
Битовое «НЕ»	NOT
Битовое «И»	AND
Битовое «ИЛИ»	OR
Битовое «ИСКЛЮЧАЮЩЕЕ ИЛИ»	XOR
Битовый сдвиг влево	SHL
Битовый сдвиг вправо	SHR

Рассмотрим принцип действия битовых операций. Поскольку эти операции работают с целым числом как с последовательностью нулей и единиц и действуют поразрядно, достаточно указать так называемые таблицы истинности операций, поясняющие, как формируется результат. Рассмотрим эти таблицы.

x	**NOT x**
0	1
1	0

Из всех битовых операций лишь операция NOT является унарной, т.е. имеет один аргумент. Принцип действия – во всех двоичных разрядах цифра меняется на противоположную ($0 \to 1$; $1 \to 0$).

x1	**x2**	**x1 AND x2**	**x1 OR x2**	**x1 XOR x2**
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

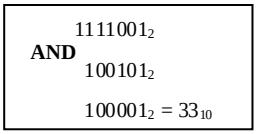
Смысл использования операций вытекает из приведенной таблицы. Операция AND возвращает 1 в очередном разряде числа тогда и только тогда, когда оба аргумента содержат в этом разряде 1. Операция OR возвращает 0 в очередном разряде числа тогда и только тогда, когда оба аргумента содержат в этом разряде 0. Операция XOR возвращает 1 в очередном разряде числа тогда и только тогда, когда аргументы содержат в этом разряде разные двоичные цифры.

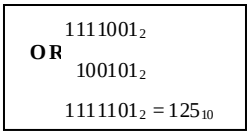
Операции AND и OR чаще всего используются для того, чтобы установить (сделать равным 1), сбросить (сделать равным нулю) или протестировать (узнать значение) какой-то бит в числе. Операция XOR используется для разных «трюков», обычно связанных с кодированием.

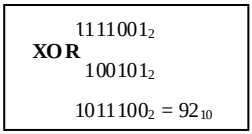
Рассмотрим некоторые примеры.

Возьмем числа 121 и 37.

$121_{10} = 1111001_2$
$37_{10} = 100101_2$







А теперь еще один пример на операцию XOR.

```
var
   a, b: Integer;
begin
   Write('Input numbers: ');
   Readln(a, b);
   a := a xor b;
   b := b xor a;
   a := a xor b;
   Write('a = ', a, ' ', 'b = ', b);
   ReadLn;
end.
```

Как вы думаете, что в ней происходит? Если запустить программу на исполнение, значения переменных a и b волшебным образом меняются местами без использования дополнительной памяти.

Операции SHL и SHR предназначены для быстрого осуществления операций умножения и деления на степень двойки. Так, x SHL 3 эквивалентно умножению на 2^3, т.е. на 8, а x SHR 3 – делению нацело на 8. Заметим, что сдвиговые операции работают существенно быстрее «обычных» операций умножения и деления.

3.5.2. Работа с вещественными числами. Для представления вещественных чисел имеется следующий набор типов: `Real`, `Real48`, `Single`, `Double`, `Extended`, `Comp`, `Currency`.

Тип `Real` (он же `Double`) является основным, `Real48` оставлен для обратной совместимости (соответствует типу `Real`[1] в ранних версиях Object Pascal). Кроме того, введен специальный тип данных `Currency`, оптимизированный для повышения точности финансовых расчетов.

Название типа данных	Множество значений	Значащих цифр	Размер
Real48	$-2.9 \cdot 10^{39}..1.7 \cdot 10^{38}$	11–12	6 байт
Single	$-1.5 \cdot 10^{45}..3.4 \cdot 10^{38}$	7–8	4 байта
Double	$-5.0 \cdot 10^{324}..1.7 \cdot 10^{308}$	15–16	8 байт
Real	$-5.0 \cdot 10^{324}..1.7 \cdot 10^{308}$	15–16	8 байт
Extended	$-3.6 \cdot 10^{4951}..1.1 \cdot 10^{4932}$	19–20	10 байт
Comp	$-2^{63}+1..2^{63}-1$	19–20	8 байт
Currency	–922337203685477.5808.. 922337203685477.5807	19–20	8 байт

Многообразие вещественных типов также вызвано соображениями повышения эффективности программы – возможность использования наиболее подходящих типов, стремление обеспечить максимальную точность вычислений.

Константы вещественных типов могут быть записаны двумя способами: с фиксированной точкой и в так называемой экспоненциальной форме, т.е. с порядком.

Константа с фиксированной точкой – обычная запись числа (со знаком или без него) с использованием точки для отделения дробной части. Например, -23.345 или 3.14.

Для компактной записи очень больших или очень маленьких по абсолютной величине чисел, т.е. содержащих много значащих нулей до или после точки, используется *запись с порядком* (*в экспоненциальной форме*). Например, вместо числа 12300000 можно записать константу `1.23e+7`. Конструкция `e+7`, записываемая после числа без пробела, указывает, что значение, расположенное до символа порядка `e` (в соответствии с общими правилами можно использовать и большую букву `E`), должно быть умножено на 10 в степени

[1] Любите ли вы рыбий жир? Нет? Тогда не удивляйтесь тому, что процессор не любит числа, имеющие размер в байтах, не являющийся степенью двойки. Определенно, 6 байт – это не то, о чем мечтает CPU.

7. Отметим, что эта запись не означает операции, т.е. преобразование константы выполняется во время компиляции, а не исполнения программы. Максимальные и минимальные значения порядков указаны в вышеприведенной таблице вещественных типов.

В языке Pascal имеются следующие *арифметические операции* над вещественными числами:

Наименование	Обозначение
Сложение	+
Вычитание	–
Умножение	*
Деление	/

Заметим, что как для целых типов, так и для вещественных не определена операция возведения в степень, которая в некоторых других языках имеет свое специальное обозначение (например, «**» в Fortran или « ^ » в Basic). Дело в том, что операция возведения в степень не реализована аппаратно, то есть как машинная команда, а представляет собой некоторый алгоритм вычислений. Исходя из соображений эффективности, программисту предлагается самому реализовать этот алгоритм.

3.5.3. Логика и логический тип данных. Рассмотрим следующее утверждение, сделанное неким студентом: «У нас в группе 5 отличников». Как отнестись к его истинности? Во-первых, с общих позиций понятно, что утверждение может быть как истинным, так и ложным. Во-вторых, сказать что-то определенное можно лишь обладая достоверной информацией. В данном случае в качестве таковой могут выступать данные деканата об успеваемости. То есть истинность этого утверждения, с одной стороны, проверяема, с другой – неоднозначна, в отличие от большинства утверждений математических. Правда, и с ними не все так просто. Чуть дальше мы обсудим эту ситуацию на одном общеизвестном примере.

С точки зрения написания программ нас в данный момент интересуют утверждения, истинность которых зависит от некоторых условий. Самое очевидное действие, связанное с такими утверждениями, – проверка. Язык программирования, конечно же, должен содержать для этого необходимые средства. Каковы они и как ими пользоваться, мы обсудим в следующей главе, а пока рассмотрим, как описать результат такой проверки.

Для работы с такими значениями многие языки программирования предоставляют специальный тип данных – *логический тип*. В языке Pascal этот тип называется `Boolean`. Логический тип имеет только два

возможных значения, представленных предопределенными идентификаторами: **True** – «истина» и **False** – «ложь».

Рассмотрим теперь следующее утверждение: «У нас в группе 5 отличников и 4 троечника». Нетрудно видеть, что эта фраза совмещает в себе два утверждения:

- «У нас в группе 5 отличников»;
- «У нас в группе 4 троечника».

Оба утверждения объединены операцией «и».

Для того чтобы обрабатывать подобные соотношения, объединять утверждения в составные конструкции, в языке необходимы специальные *логические операции*.

Наименование	Обозначение
НЕ (отрицание)	NOT
И (конъюнкция)	AND
ИЛИ (дизъюнкция)	OR
ИСКЛЮЧАЮЩЕЕ ИЛИ	XOR

Важно не путать эти операции с рассмотренными ранее битовыми операциями. Если те представляли целочисленные аргументы как последовательность двоичных разрядов и работали *поразрядно*, то логические операции применимы только к двум возможным значениям операндов – «Истина» и «Ложь».

Итак, *унарная* (то есть имеющая один аргумент) операция NOT дает результат, противоположный аргументу по значению. Все остальные логические операции имеют по два аргумента, то есть являются *бинарными*. Результат операции AND есть **True** тогда и только тогда, когда оба аргумента имеют значение **True**. Операция OR возвращает **False** тогда и только тогда, когда оба аргумента имеют значение **False**. Операция XOR возвращает **True**, если значения аргументов противоположны.

В дополнение к логическим языки программирования обычно предоставляют *операции отношения*, которые в качестве аргументов могут принимать значения произвольного порядкового типа (*порядковым* является любой тип, элементы которого можно занумеровать), результат же таких операций принадлежит логическому типу. Операции отношения в языке Pascal выглядят следующим образом:

Наименование	Обозначение
Больше	>

Меньше	<
Больше или равно	>=
Меньше или равно	<=
Равно	=
Не равно	<>

Важно, что сравниваться могут не только числа, но и значения других порядковых типов. Так, например, тип `Boolean` является порядковым, а выражение **`False`** < **`True`** истинным.

Отметим также, что для совместимости с программным обеспечением, реализованным при помощи других систем программирования, язык Object Pascal содержит логические типы данных `ByteBool`, `WordBool`, `LongBool`.

В заключение раздела небольшая логическая задача.

Рассмотрим следующее утверждение: «2 × 2 = 4». Что мы можем сказать об его истинности? Первое, что приходит в голову любому, кто хотя бы слышал о таблице умножения, – конечно же, утверждение истинно! Однако не все так просто. В этой формулировке неявно подразумевается, что речь идет о всем привычной десятичной системе счисления, где данное утверждение трудно оспорить. Но мы-то с вами теперь более грамотны и знаем, что существуют системы счисления и с основаниями, отличными от десяти, например с основанием три. В этой системе, как не трудно догадаться, отсутствует цифра 4, а значит, результат умножения будет следующий: «$2 \times 2 = 11_3$». Таким образом, на самом деле истинность указанного выше утверждения не так уж и бесспорна.

Конечно, можно возразить, что результат операции умножения есть одно и то же число, а от системы счисления зависит лишь форма его представления. Это тоже верно. Однако представьте себе, что вам потребовалось создать программу-калькулятор, способную работать с числами в любой позиционной системе счисления. Можно ли будет в качестве теста к этой программе использовать утверждение, указанное выше? Очевидно, нет.

Мораль всего сказанного – к любым даже самым очевидным с обыденной точки зрения утверждениям при разработке программ нужно подходить критически.

3.5.4. Символьная информация и символьный тип данных. Константами этого типа, имеющего имя `Char`, являются символы, обрабатываемые компьютером в стандартном текстовом режиме работы. Это набор из 256 символов, представленных в памяти в коде ASCII (American Standard Code for Information Interchange). Код символа занимает

1 байт – 1 байт = = 8 бит, следовательно, имеем $2^8 = 256$ различных комбинаций. В их число входят буквы (прописные и строчные), цифры и большой набор специальных знаков, например так называемые символы псевдографики. Входят в число допустимых символов и русские буквы, разумеется, при наличии на компьютере специального аппаратного или программного обеспечения перекодировки стандартной таблицы символов. Заметим, что все символы упорядочены по значениям их кодов. При этом буквам присвоены такие коды, чтобы сохранялся алфавитный порядок.

Константы символьного типа могут быть записаны двумя способами. Первый способ – задание в явном виде в апострофах, например, `'a'` или `'$'`. Данный способ при вводе информации, очевидно, годится лишь для символов, представленных на клавиатуре.

Второй способ задания символьной константы – это указание ее десятичного кода, перед которым записывается специальный знак #. Например, #21. В языке Pascal отсутствуют специальные операции обработки символов, однако символьный тип является порядковым, и к константам и переменным этого типа применимы операции сравнения.

Заметим, что 256 символов – не панацея. Как быть в Китае и Японии? Для того чтобы иметь возможность представления 65536 различных символов, был принят стандарт Unicode, представленный в Object Pascal типом данных `WideChar`.

3.5.5. Строковая информация и строковый тип данных. Обработка информации, представленной в строковом виде, не менее распространена, чем численная. Мы имеем дело с названиями улиц, днями недели, именами и т.д. Конечно же, языки программирования должны предоставлять соответствующий аппарат для такого рода данных.

В первом приближении строка есть последовательность символов. Язык Object Pascal содержит несколько типов данных, позволяющих представлять строковую информацию. Самый простой из них мы рассмотрим сейчас, остальные – в главе 9.

Рассмотрим следующие объявления:

```
const
   TestStr = 'Это пример строки';
var
   s: ShortString; { Это переменная – строка              }
   s1: String[30]; { Это тоже строка, ограниченной длины }
```

Первое объявление задает константу строкового типа. Как видите, в этом случае значение константы заключается в апострофы. Длина заданной таким способом строки не может превышать 255 символов. Вытекает это из

того факта, что для хранения длины используется один байт, располагаемый в самом начале строки (поэтому общий объем выделяемой памяти не может быть больше 256 байт).

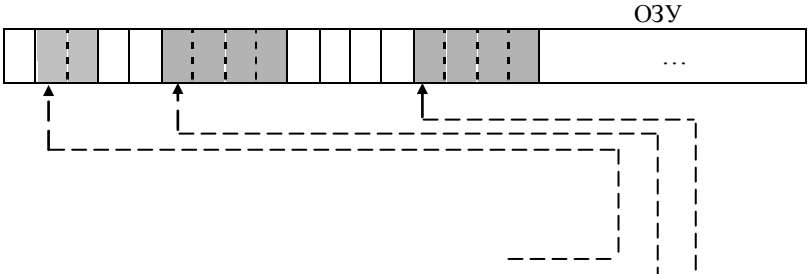
Объявленная далее переменная `s` также может содержать строку до 255 символов длиной, однако, в отличие от константы, в памяти для нее выделено в точности 256 байт и размер этот от длины строки никак не зависит.

В случае когда заранее известно, что строки, с которыми придется работать в программе, не превышают по длине некоторого числа, меньшего 255, можно сэкономить и объявлять переменные так, как это сделано с переменной `s1`, указывая после имени типа максимальную длину строки в квадратных скобках. Заметьте, что при объявлении `s1` использовано название типа `String`. Дело в том, что в Object Pascal тип `ShortString` оставлен для обратной совместимости с программами, написанными на основе предыдущих версий языка. Однако устройство типа `String` существенно более сложно, поэтому пока мы ограничимся рассмотренным вариантом.

Операции над строками реализованы в виде стандартных подпрограмм. Мы также рассмотрим их в главе 9. А здесь познакомимся с одной часто употребляемой операцией, которая к тому же имеет весьма простой вид. Это операция объединения строк, называемая еще *конкатенацией*. Ее результат – строка, объединяющая строки-аргументы в порядке их следования. Обозначается эта операция знаком «+». Например, результатом операции `'Object ' + 'Pascal'` будет строка `'Object Pascal'`.

4. Понятие переменной

В целом смысл понятия переменной совпадает с его общепринятой математической интерпретацией как символического имени, которому в разные моменты времени могут быть приписаны различные значения. Вместе с тем с программистской точки зрения требуются уточнения. *Переменная* есть символическое (в виде идентификатора) обозначение некоторой области памяти, в которую в результате выполнения операторов программы помещаются значения.



Переменные	
Идентификатор	Адрес
A	•
Number	•
TheValue	•

Рис. 4.2. Переменные – поименованные области памяти

Переменная характеризуется типом данных, значения которого она может хранить. Тип данных, в свою очередь, определяет размер области памяти, которая должна быть отведена под переменную, и способ интерпретации содержимого этой памяти. Важной особенностью многих языков программирования, в том числе и языка Pascal, является правило, согласно которому переменная в *области своего действия* может принадлежать только одному типу, т.е., например, одной и той же переменной не может быть приписано вещественное и строковое значение в процессе работы программы. В этом правиле требуют пояснения слова об области действия переменной. Коротко говоря, область действия – это блок. Более подробно этот вопрос мы рассмотрим в главе, посвященной модульному программированию.

Как мы уже обсуждали ранее, переменные объявляются в разделе объявлений, начинающемся с ключевого слова **var** (от английского variable – переменная). Объявление имеет следующий общий вид:

```
var
  <Список идентификаторов>: <Тип данных>;
```

Список идентификаторов есть список из имен переменных, которые мы хотим объявить как переменные одного типа. Сам тип указывается после двоеточия.

Тип данных – это, например, одно из тех имен встроенных типов, которые мы рассмотрели выше. Приведем примеры объявления переменных предопределенных типов:

```
var
  a, b, c: Real;
  Page_Number: Word;
  FileName: String[80];
  Key: Char;
```

5. Понятие константы

Мы уже знаем, как записываются константы встроенных типов. Однако в добавление к стандартному способу представления в языке Pascal имеется возможность вводить псевдонимы для констант любых типов. Эта замечательная возможность, во-первых, облегчает модификацию программы, а во-вторых, делает текст программы более понятным.

Объявления констант производятся в разделе, начинающемся с зарезервированного слова **const**. Конструкция объявления константы имеет следующий общий вид:

```
const
  <Имя константы> = <Значение>;
```

Например,

```
const
  Ln10 = 2.302585;
  SecondsInDay = 24 * 60 * 60;
  Done = True;
  Path = 'C:\MyPrograms\';
  ВМК = 'Факультет ВМК';
  SizeOfTable = 50;
```

Как видим, значение константы может задаваться выражением, содержащим другие константы, как явные, так и определенные ранее, а также некоторые допустимые функции от констант. Основное правило составления таких выражений (в том числе и использования в них функций) – компилятор должен иметь все необходимое для вычисления значения константы.

Значение константы во время выполнения программы изменено быть не может.

Использование именованных констант вместо явных обеспечивает ряд преимуществ. Текст программы становится более ясным, если, например, вместо непонятно откуда взятых чисел 2.302585 или 86400 будут стоять поясняющие идентификаторы `Ln10` и `SecondsInDay`, имена которых ясно говорят, что же имеется в виду. Также текст программы может быть сокращен – без особой потери ясности, если вместо длинной часто встречающейся строковой константы записывать ее обозначение (смотри пример с константой `ВМК`).

Другое достоинство использования констант связано с существенным облегчением модификации программы. Допустим, некоторая программа ориентирована на обработку таблиц с максимальным числом строк, равным 50. Эта константа многократно встречается в программе, например в алгоритмах просмотра таблицы, контроля ввода и т.п. Если по какой-либо причине необходимо сменить максимально допустимый размер таблицы, то поиск явной константы 50 в программе может вылиться в утомительное

занятие. При этом не исключены и ошибочные замены, так как число 50 могло быть использовано и в других целях. Очевидно, при использовании в тексте программы вместо явной константы 50 именованной константы `SizeOfTable` достаточно будет сменить приписанное ей значение.

6. Типизированные константы и инициализация переменных

Описывая в предыдущем разделе способ задания именованных констант, мы опустили один довольно важный момент. Как компилятор «разбирается», какой тип назначить указанной нами константе? Для встроенных типов данных этот вопрос может быть решен автоматически. Для целочисленных констант выбирается самый «младший» подходящий тип. Для вещественных – тип `Double` или `Extended` в зависимости от числа указанных цифр после запятой и величины константы. Однако бывают ситуации, когда программисту требуется явным образом указать тип константы. В этом случае язык Object Pascal предоставляет возможность использовать *типизированные константы*:

```
const
   Ln10: Single = 2.302585;
   SecondsInDay: Integer = 24 * 60 * 60;
```

Как видите, формат похож на объявление переменной с той лишь разницей, что после указания типа ставится знак « = » и задается значение константы. Наиболее полезны типизированные константы для типов данных, создаваемых программистом, о чем пойдет речь в главе 6.

Аналогичным способом в языке Object Pascal можно инициализировать переменные начальными значениями непосредственно при объявлении:

```
var
   Square: Single = 0;
   NumOfElements: Integer = 100;
```

В отличие от типизированных констант[1] значения инициализированных переменных, конечно же, можно изменить в теле программы.

```
begin
   Square := a * b;
```

[1] Необходимо отметить, что в версии языка Pascal, на которой основана среда разработки Borland Pascal 7.0, типизированные константы фактически являлись переменными, то есть их значения можно было менять, а инициализировать переменные при объявлении было нельзя.

7. Оператор присваивания и выражения

Одним из существенных качеств любой программы является ее общность, то есть способность выполнять заложенные в нее действия на разных наборах данных. Из этой посылки с неизбежностью вытекает тот факт, что основной объект любой программы – переменная, способная принимать и хранить различные значения. В силу важности этого понятия повторим еще раз данное выше определение: *переменная есть символическое обозначение места в памяти*. Естественно, любой язык программирования должен содержать операцию установки требуемого значения в это самое место. Такая операция называется *оператор присваивания*.

В языке Pascal оператор присваивания имеет вид:

```
<переменная> := <Выражение>;
```

где « := » – знак присваивания.

Примеры оператора присваивания:

```
a := 5;
x1 := (b + sqrt(sqr(b) - 4 * a * c)) / (a + a);
Str := 'Это ' + 'строковое ' + 'выражение';
IsGoodStudent := AvgMark >= 4;
```

Пользуясь имеющимися в языке операциями, программист может составлять формулы или *выражения* над значениями типов данных. Существуют общие естественные правила построения выражений, которые могут включать в себя операции над константами, переменными и значениями, заданными функциями. Для указания порядка действий, отличного от порядка, определяемого приоритетами операций, используются круглые скобки.

Для лучшего понимания принципа работы оператора присваивания рассмотрим еще один вариант его записи, часто встречающийся в литературе:

```
<LValue> := <RValue>;
```

Здесь LValue – левая часть оператора присваивания – должна обязательно указывать на какую-то область памяти. Так, например, неправильны и порождают сообщения об ошибке следующие примеры:

```
5 := a;      { Число 5 не связано с областью памяти! }
a + b := 7; { Результат выражения «a+b» не связан   }
             { с областью памяти!                    }
```

Надеемся, что необходимость привязки `LValue` к области памяти для вас очевидна. В самом деле, иначе было бы непонятно, куда же собственно помещать результат выполнения присваивания.

`RValue` – правая часть оператора присваивания – в отличие от левой может содержать любые допустимые в языке выражения.

Подводя итог, укажем алгоритм выполнения оператора присваивания:

1. Вычисляется значение выражения, стоящего в правой части (`RValue`).
2. Определяется адрес оперативной памяти, по которому необходимо записать значение (`LValue`).
3. Производится запись значения, вычисленного на шаге 1, по адресу, определенному на шаге 2, в соответствии с типом данных переменной (`LValue`).

8. Преобразования встроенных типов данных

Допустим, сделаны следующие объявления переменных:

```
var
   x, a, d: Real;
   i, j: Integer;
```

Рассмотрим оператор присваивания:

```
x := a + (i div j) / d;
```

Как видим, правая часть оператора состоит из выражений разных типов. Допустима ли такая запись? Для данного примера ответ положителен. В общем же случае ответ на этот вопрос не может быть дан однозначно и требует обстоятельного рассмотрения. В данном разделе мы сделаем первые замечания по этой проблеме и разберем случай предопределенных типов данных.

Итак, как нам уже известно, тип данных определяет, помимо прочего, способ представления значений в оперативной памяти. Различия в представлении, в свою очередь, неизбежно ведут к необходимости каждому типу данных «иметь» собственные команды обработки значений, машинные или программно реализованные. Таким образом, если мы выполняем целочисленную операцию деления (**div**) целого числа на вещественное, то команда деления будет интерпретировать код

вещественного числа как код целого, следовательно, результат операции окажется весьма далек от ожидаемого. Конечно, на практике выполнить такую операцию не удастся, поскольку компилятор выдаст сообщение об ошибке вида: «Тип операнда не соответствует операции». В свою очередь, такая диагностика возможна благодаря наличию типизации в языке Pascal.

Попытаемся теперь сложить целое и вещественное числа. Вам ничего не кажется подозрительным? Как мы указали выше, операция сложения имеет одно и то же обозначение как для целых, так и для вещественных аргументов: «+». Как же ее будет рассматривать компилятор в данном случае? Очевидно, что операцию нельзя выполнять как сложение целых чисел – результат окажется неверным. Возможно то же решение, что и в случае с делением, – выдать сообщение об ошибке. Однако это противоречит естественному порядку, принятому в математике, – мы выполняем операции сложения чисел, не задумываясь о соответствии типов. В данном случае разумным представляется преобразовать целое число к вещественному и выполнить сложение над вещественными аргументами. Именно так и делается на практике: компилятор осуществляет автоматическое преобразование или, как еще говорят, приведение типов и выбирает соответствующую команду обработки.

В общем случае преобразование выполняется к «старшим» типам, то есть к вещественным и с большим диапазоном значений. Таким образом, в выражении могут смешиваться целые и вещественные типы всех разновидностей, но программист в этом случае должен быть очень внимателен и понимать, в каком порядке будут осуществлять преобразования и собственно выполнение операций, иначе результаты расчетов могут его сильно удивить.

Отметим один существенный факт: в операторе присваивания в правой части может стоять целое выражение, а в левой вещественная переменная, но не наоборот. Для приведения вещественного аргумента к целому типу можно воспользоваться функциями явного преобразования: `Round(x)` и `Trunc(x)`. Обе функции преобразуют вещественное число к типу `LongInt`, причем первая округляет число, а вторая просто отбрасывает дробную часть.

Для нечисловых типов данных аппарат автоматического преобразования типов отсутствует, что, конечно, вполне понятно. В выражениях нельзя смешивать числа, логические и символьные значения, т.е., например, запись `x := 2 + '2'` недопустима.

С другой стороны, существует возможность явного преобразования символов и строк в числа и наоборот. Для символов это функция `Ord()`, которая возвращает десятичный порядковый номер (код ASCII) символа, и

функция `Chr()`, которая по десятичному коду выдает соответствующий символ.

Текстовые строки, которые имеют вид, совпадающий с синтаксическим определением числовой константы, преобразуются с помощью процедур `Val` и `Str`, включенных в библиотеку `System`. Процедуры, так же как функции, являются заранее разработанными библиотечными программами и отличаются от функций способом обращения к ним. Оператор вызова процедуры в языке Object Pascal записывается очень просто. Это имя процедуры, за которым в круглых скобках следует список параметров. В дополнение к рассмотренным подпрограммам, ставшим стандартными со времен Borland Pascal, язык Object Pascal содержит ряд новых процедур и функций: `FloatToStr`, `StrToFloat`, `DateToStr` и т.д. Их описание может быть без труда найдено в справочной системе, здесь мы его дублировать не будем.

9. Некоторые стандартные математические функции

Обычно языки программирования содержат лишь необходимый минимум выразительных средств. В дополнение к ним создатели сред разработки добавляют библиотеки функций, содержащие эффективные реализации некоторого множества часто используемых операций. В их число входят, например, подпрограммы вычисления элементарных математических функций, подпрограммы обработки строк и другие.

В данном пункте мы приведем некоторые стандартные математические подпрограммы, входящие в состав библиотеки System. Особенность этой библиотеки состоит в том, что она всегда автоматически подключается редактором связей к собираемой программе пользователя, т.е. использование подпрограмм из нее не требует подключения в секции uses.

Функция	Описание
`Abs(x)`	Взятие модуля
`Sqr(x)`	Возведение в квадрат
`Sqrt(x)`	Извлечение квадратного корня
`Exp(x)`	e^x
`Ln(x)`	Логарифм натуральный от x
`Sin(x)`	Синус
`Cos(x)`	Косинус
`ArcTan(x)`	Арктангенс
`Int(x)`	Целая часть
`Frac(x)`	Дробная часть

Заметим, что в число функций не входит, например, функция тангенс и ряд других элементарных функций. Их значения вычисляются с помощью формул над приведенными в таблице функциями. В частности, показательная функция (a^x) может быть реализована как `Exp(Ln(a)*x)`.

Система программирования Borland Delphi содержит модуль Math, реализующий многие другие математические функции (арифметические, тригонометрические, финансовые, статистические). Эти и некоторые другие стандартные подпрограммы мы будем рассматривать по мере необходимости.

10. Простейшие средства ввода и вывода информации. Стандартный ввод-вывод в консольном приложении

Программирование ввода и вывода информации, а в общем смысле организация взаимодействия с пользователем – одна из важнейших задач для разработчика. Уже в первом примере книги мы видели, что число операторов, обеспечивающих обмен информацией, превышало число операторов расчетных. Это, конечно, не всегда так, но программа, предназначенная для коммерческого использования, как правило, содержит весьма емкий блок, реализующий интерфейс с пользователем. В настоящее время подавляющее большинство таких программ используют интерфейс графический (GUI – graphic user interface), основанный на понятии окна, из-за чего сами программы обычно называются оконными приложениями. Построение подобного интерфейса представляет собой тему для отдельной книги, таких книг немало (откройте, например, любую книгу по системе визуального программирования Borland Delphi [20, 25, 26]). Однако существует класс задач, в которых возможности графического интерфейса и оконных приложений не нужны. Так, например, к этому классу относятся большинство расчетных программ, системные утилиты. В таких случаях в качестве устройства вывода используется текстовая консоль. В данном разделе мы познакомимся с тем, как программируется простейший ввод/вывод в текстовом режиме в консольных приложениях. При этом мы узнаем, как запрашивать от пользователя данные и выводить на экран результаты. Желающие организовывать полноценный интерфейс в текстовом режиме с использованием различных цветов, очисткой экрана,

позиционированием курсора и т.д. могут обратиться к справочной системе MSDN[1].

Подпрограмма ввода с клавиатуры имеет вид:

```
Read[(<список переменных>)];
```

или

```
Readln[(<список переменных>)];
```

При этом `список переменных` может содержать переменные любых стандартных типов языка Pascal.

Данная подпрограмма является весьма сложной и обеспечивает:

- отображение нажимаемых в процессе ввода клавиш-символов на экране дисплея;
- синтаксический контроль вводимой информации;
- преобразование данных из символьного (внешнего) представления во внутренний код, соответствующий типу записанной в списке ввода переменной.

Схема работы `Read` следующая:

- программа переходит в режим ожидания ввода с клавиатуры;
- пользователь осуществляет ввод информации, отделяя значения переменных пробелами или нажатием клавиши Enter;
- признаком завершения ввода является нажатие клавиши Enter при одновременном исчерпании списка ввода (лишние значения будут проигнорированы);
- начинается обработка введенной информации: преобразование из символьного представления к типу данных соответствующей переменной в списке ввода;
- если введенные данные некорректны, например производится попытка ввести в целую переменную символы, недопустимые в представлении целых чисел, работа всей программы аварийно завершается и выдается сообщение об ошибке времени исполнения (например, «Invalid numeric input» – «Неверный численный ввод»).

Необходимо обратить внимание на один существенный момент в работе подпрограммы `Read` – последний символ Enter, которым заканчивается ввод информации, `Read` *не считывает*. Для этого предназначена вторая форма подпрограммы `Readln`.

В соответствии с представленным выше форматом подпрограммы `Read` и `Readln` могут не иметь списка ввода. При этом `Read` фактически не

[1] MSDN – Microsoft Developer Network – библиотека справочной информации по продуктам Microsoft (www.msdn.com).

выполнит никаких действий, а Readln будет ожидать нажатия клавиши Enter (если она не была нажата ранее и не считана).

Подпрограмма вывода на дисплей имеет вид:

```
Write[(<список вывода>)]; или Writeln[(<список вывода>)];
```

В список вывода входят *выражения*, разделенные запятыми. Значения выражений преобразуются к строковому виду и выводятся на экран. Имеется возможность форматирования выводимых значений, такая же, как и в процедуре Str. Например, предложение

```
Write(Sin(Pi/4):5:3);
```

выдаст результат в виде 0.707, в то время как предложение

```
Write(Sin(Pi/4));
```

выдаст на экран число в стандартной экспоненциальной форме 7.0710678119E-01.

Если программист ошибся в выборе формата и, например, указал общее число позиций, меньшее, чем число значащих цифр, то программа сама исправит положение и напечатает число в стольких позициях, сколько необходимо для размещения числа.

Подпрограмма Writeln помимо вывода значений из списка вывода обеспечивает перевод курсора в начало следующей строки.

Заметим, что значения всех рассмотренных нами встроенных типов могут быть введены и выведены указанными подпрограммами. Исключение составляет ввод булевских значений. Read не предназначен для чтения констант **True** и **False**. Сообщение об ошибке будет выдано еще на этапе компиляции. В то же время Write выводит эти константы в строковом виде.

Рассмотрим примеры.

```
{ ================================================= }
{ Пример 4.1 }
{ Вывод на дисплей десятичного кода ASCII требуемого символа }
Program DisplayCode;

{$APPTYPE CONSOLE}
var
   Symbol: Char;
begin
   Write('Символ: ');
   Readln(Symbol);
   Writeln('Код: ', Ord(Symbol));
   Readln;
```

```
end.
{ ================================================ }
```

Эта программа запрашивает ввод символа, читает его и печатает код. На экране диалог выглядит следующим образом:

```
Символ: k
Код: 107
```

```
{ ================================================ }
{ Пример 4.2 }
{ Значения элементарных функций }
Program Functions;

{$APPTYPE CONSOLE}

var
   x: Real;
begin
   Write('Аргумент? ');
   Readln(x);
   Writeln('Sin(',x:5:3,') = ', Sin(x):5:3);
   Writeln('Cos(',x:5:3,') = ', Cos(x):5:3);
   Writeln('Exp(',x:5:3,') = ', Exp(x):5:3);
   Readln;
end.
{ ================================================ }
```

Результат работы программы:

```
Аргумент? 3.45
Sin(3.450) = -0.304
Cos(3.450) = -0.953
Exp(3.450) = 31.500
```

11. Выводы

Нам представляется, что данная глава очень важна не только с практической, но и с теоретической точки зрения. Тем из вас, кто выберет программирование в качестве своей профессии, предстоит изучить многие языки программирования, познакомиться с новыми средствами и технологиями и, быть может, даже увековечить свое имя в зале славы их разработчиков. Совершенно ясно, что изучение нового материала не должно занимать месяцы, у вас просто не будет так много времени в нашем динамично меняющемся мире. Содержание главы представляет собой тот

базис, те фундаментальные понятия и принципы, на основе которых, как правило, и строятся языки программирования.

Литература

1. Богатырев Р. Летопись языков Паскаль // Мир ПК. –№ 4.–2001.

2. Богатырев Р. От Паскаля к языку Zonnon: реализация новых идей на платформе .NET // Мир ПК.–2003.–№ 9.

3. Большая советская энциклопедия. Издание третье. – М.: Советская энциклопедия, 1970.

4. Брэдли Д. Программирование на языке ассемблера для персональной ЭВМ фирмы IBM.–М.: Радио и связь, 1988.

5. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++. Второе издание. – Бином, 1998.

6. Касперски К. Техника оптимизации программ. Эффективное использование памяти.– СПб.: БХВ-Петербург, 2003.

7. Кетков А., Кетков Ю. Практика программирования: Бейсик, Си, Паскаль. Самоучитель.–СПб.: БХВ-Петербург, 2001.

8. Кетков А., Кетков Ю. Практика программирования: Visual Basic, C++ Builder, Delphi. Самоучитель.–СПб.: БХВ-Петербург, 2002.

9. Кнут Д.Э. Искусство программирования. Т. 1. Основные алгоритмы.–М.:Вильямс, 2000.

10. Кнут Д.Э. Искусство программирования. Т. 3. Сортировка и поиск.–М.:Вильямс, 2004.

11. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы. Построение и анализ.–М.:МЦНМО, 1999.

12. Кэнту М. Delphi 7 для профессионалов.–Спб.:Питер, 2004.

13. Лингер Р., Миллс Х., Уитт Б. Теория и практика структурного программирования.–М.: Мир, 1982.

14. Модернизация и ремонт персонального компьютера.– [http://www.allcompinfo.com]

15. Новичков А. Система генерации проектной документации Rational SoDA.– [http://www.citforum.ru/programming/digest/soda.shtml]

16. Олифер В.Г., Олифер Н.А. Сетевые операционные системы.–СПб.: Питер, 2002.

17. Рихтер Дж. Windows для профессионалов: создание эффективных Win32 приложений с учетом специфики 64-разрядной версии Windows / Пер. с англ.– 4-е изд.–СПб.: Питер; М.: Издательско-торговый дом «Русская Редакция», 2001.

18. Служба тематических толковых словарей. – [http://www.glossary.ru/]

19. Стивенс Р. Delphi. Готовые алгоритмы.– Спб.:Питер, 2004.

20. Тейксейра С., Пачеко К. Borland Delphi 6. Руководство разработчика. –Вильямс, 2002.

21. Шень А. Программирование: теоремы и задачи.–М.:МЦНМО, 2004.

22. Юров В. Assembler.–СПб.: Питер, 2002.

23. Cantu M. Essential Delphi.– [www.marcocantu.com/edelphi]

24. Cantu M. Essential Pascal.– [www.marcocantu.com/epascal]

25. Cantu M. Mastering Borland Delphi 2005.–Sybex, 2005.

26. Cantu M. Mastering Delphi 7.–Sybex, 2003.

27. Clark R., Koehler S. The UCSD Pascal Handbook.–Prentice-Hall, 1982.

28. Dahl O., Dijkstra E., Hoare C.A.R. Structured Programming.–London, England: Academic Press, 1972.

29. Gutknecht J., Zueff E. Zonnon Language Report. Draft.–ETH Zurich, June 2003.

30. Gutknecht J., Zueff E. Zonnon for .NET, A Language and Compiler Experiment. // Computer Systems Institute, ETH Zürich, Switzerland. JMLC Conference.–August 2003.

31. Gutknecht J. Zonnon: A .NET Language Beyond C# // Moscow: Microsoft Conference, June 15-17, 2003.

32. Gutknecht J., Wirth N. The Oberon System // Software – Practice and Experience.–Vol.19, № 9.–1989.–p.857-893.

33. Jacobson I., Christerson M., Jonsson P., Overgaard G. Object-oriented Software Engineering.–Workingham, England: Addison-Wesley Publishing Company, 1992.

34. Jensen K., Wirth N. PASCAL – User Manual and Report, ISO Pascal Standard.–1974.

35. Intel Architecture Software Developer's Manual. Volume 1: Basic Architecture.– Intel Corporation: 1997.

36. Meyer B. Object-oriented Software Construction.–Prentice Hall, 1988.

37. Rumbaugh H., Blaha M., Premarlani W., Eddy F., Lorensen W. Object-Oriented Modeling and Design.–Prentice-Hall, 1995.

38. Sedgewick R. Algorithms.–Reading, MA: Addison-Wesley, 1983.

39. Tesler L. Object Pascal Report. // Structured Language World, Vol.9, No.3.– 1985.–p. 10-14.

40. Thiriez H. Modelling of an interactive scheduling system in a complex environment // European Jornal of Operational Research. –1991.– №50.–p.37-47.

41. Wirth N. Program Development by Stepwise Refinement // Communications of the ACM vol.26(1).– January 1983.

42. Wirth N. The Programming Language Pascal // Acta Informatica, 1.–1971.–p. 35-63.

43. Wirth N. Programming in Modula-2.–Springer-Verlag, 1982.

44. Wirth N. The programming language Oberon // Software – Practice and Experience, Vol.18, № 7.–1988.–p.671-690.

45. Wirth N. Programming in Modula-2.–Springer, 1974.

46. Wirth N. Pascal and its Successors // Conference on Computer Pioneers.–Bonn, 2001.