О языках программирования вообще

Профессор. Давайте сначала рассмотрим язык программирования не в узкопрофессиональном, а в более широком плане. Другими словами, я предлагаю вам сначала ознакомиться с самыми общими категориями, так или иначе практически воплотившимися во всех современных языках программирования. Без их понимания невозможно приступить к изучению ни первого в вашей жизни, ни последующих конкретных языков. Более того, не овладев общими понятиями, нельзя эффективно продвигаться в решении конкретных задач, усваивать уже устоявшиеся, а также новые технологии программирования.

Все языки программирования включают в себя по меньшей мере следующие понятия:

- переменные и типы данных;
- массивы данных;
- функции;
- объекты;
- операторы;
- выражения.

Эти краеугольные составляющие могут воплощаться или, другими словами, поддерживаться различными языками по-разному. Далее мы попытаемся уяснить собственно базовые понятия, но прежде обратимся к аналогиям с естественными языками.

Естественные языки базируются на таких категориях, как имя существительное, имя прилагательное, глагол и т. п. Слова, которые мы встречаем в обычной живой речи, принадлежат к той или иной категории языка. Так, слово "стол" — имя существительное, "красный" — прилагательное, а "писать" — глагол.

Категории естественного языка воплощаются в речи, исполняя ту или иную роль члена предложения (выражения языка), например, подлежащего, сказуемого, определения, дополнения, обстоятельства и т. п.

При этом исходные слова адаптируются согласно правилам построения предложений. Так, существительные и прилагательные склоняются по падежам, а глаголы спрягаются по временам. Вместе с тем слово одной и той же категории в предложении может выполнять различные функции. Например, существительное может фигурировать в предложении и как подлежащее (главный член предложения), и как дополнение (второстепенный член), в зависимости от структуры предложения. Итак, слова различных категорий языка, т. е. составляющих его базу, могут выполнять различные роли в предложении, или реализации словоупотребления. Как и в естественном языке, примерно то же самое происходит и в языках программирования.

Если вы новичок, то первое, на что вы должны обратить внимание при изучении языка программирования, — как реализуются на практике перечисленные понятия. Если вы уже знакомы с каким-нибудь языком, то при освоении нового следует лишь уточнить особенности реализации базовых понятий.

2.1. Переменные и типы данных

Профессор. Важнейшим в программировании является понятие переменной, с уяснения которого и следует начать. Оно изначально возникло в математике задолго до появления первых вычислительных устройств и программ для них. В математике под переменной понимают то, что может быть заменено конкретным значением. В уравнениях, известных вам еще со школы, фигурируют переменные, обозначаемые чаще всего символом "x". Разумеется, обозначать переменные можно и другими буквами или их последовательностями (например, "s", "myvar" или "сумма"). Вот типичный пример уравнения, встречающегося в математике: $x^2 - 2 = 0$.

Здесь x — символ, вместо которого можно подставлять различные значения, но не совсем произвольные, а именно числа. При этом говорят также, что x пробегает значения из множества чисел. Заметьте, что данная переменная не может принимать значения в виде строк, составленных из букв, а также дат (например, 16.12.2007) или других данных нечислового типа. Иначе говоря, переменная в математике — это символьное обозначение некоторого числового значения, но не любого, а лишь принадлежащего определенному множеству, например, всех чисел, или только действительных, или только целых, или каких-то других. Таким образом, переменная это заместитель в некотором выражении (формуле) конкретного значения из наперед заданного множества или, как еще говорят, типа данных.

Итак, переменная может фигурировать в некотором математическом выражении. Что она там делает и каков смысл данного выражения с ее участием?

Простак. Переменная выполняет, как Вы нам напомнили, исключительно "представительские" функции. Она, так сказать, представляет некоторое, а лучше сказать, любое значение из заранее определенного множества, которое Вы, профессор, назвали типом значений.

Профессор. Совершенно верно, переменная представляет значения определенного типа. Вы достаточно аккуратно охарактеризовали понятие переменной, а потому, прошу Вас, продолжите рассуждение относительно участия переменной в выражении.

Простак. Попытаюсь, насколько смогу. В самом деле, переменная x участвует в некотором выражении вкупе с другими символами. Те, кто учился в школе, легко могут понять, что приведенное Вами, профессор, выражение констатирует тот факт, что x в квадрате за вычетом двух равно нулю. Сам собой напрашивается вопрос: а каково конкретное значение переменной, которое обеспечивает выполнение данного равенства?

Зануда. Вот именно, приведенное ранее выражение просто констатирует, что некоторое число, пока нам неизвестное, удовлетворяет математическому уравнению.

Вопрос о конкретном значении переменной *x*, удовлетворяющем данному равенству, кажется вполне естественным, он напрашивается сам собой.

Профессор. Уважаемые коллеги, столкнувшись с некоторым уравнением, вы сразу же формулируете некую математическую задачу, решение которой должны дать математики, хотя в данном случае любой школьник скажет, что только два числа, $+\sqrt{2}$ и $-\sqrt{2}$ (в десятичном выражении это 1,4142… и –1,4142…), удовлетворяют данному уравнению. Естественно, встреча с чем-то новым порождает попытку понять, к чему бы это. Если это математическое уравнение, то само собой речь пойдет о его решении, т. е. о вычислении тех значений переменной, при которых данное уравнение выполняется. Однако с точки зрения программирования дело совсем не в этом.

В математике, по крайней мере в школьной, обычно рассматривают всяческие уравнения и неравенства. При этом зачастую сложные и длинные выражения обозначают более компактными именами, т. е. вводят в обиход новые переменные. Этот прием позволяет в дальнейшем сократить математические выкладки, заменяя данное многосимвольное выражение более коротким.

В математике в договоре о новом обозначении или, другими словами, о вводе новой переменной, обычно говорят что-то в следующем духе:

"обозначим через *y* выражение $x^2 - 2$";

"пусть *y* равно $x^2 - 2$".

Как в математике, так и в программах подобные предложения записывают кратко: $y = x^2 - 1$.

Если в математике данное выражение является краткой записью соглашения о введении нового обозначения или констатацией равенства значений левой и правой частей равенства, то в программировании это команда, требующая присвоить переменной *y* значение выражения, расположенного справа от знака равенства. Чтобы эта команда могла быть выполнена компьютером, необходимо сначала вычислить значение выражения справа от знака равенства, и только потом присвоить полученное значение переменной.

Зануда. Я достаточно хорошо понял, как следует понимать в языках программирования простые выражения, например, $x = 5$ или $y = x^2 - 1$. Но тогда что означает в программировании выражение вида $x^2 - 2 = 0$? Мне непонятно, как можно левой части равенства, представляющей не просто какую-то переменную, а целое выражение, присвоить некоторое значение. Конечно, в математике такое возможно. Например, мы можем сказать так: "приравняем данное выражение нулю". А что это означает в языках программирования?

Профессор. В математике данное выражение обычно трактуется как утверждение о том, что левая часть равенства эквивалентна правой. Соответствующее предложение на естественном языке имеет изъявительное наклонение, т. е. это повествовательное предложение, утверждающее в данном случае факт равенства. В повелительном наклонении то же предложение можно сформулировать, например, так: "пусть

$x^2 - 2 = 0$". В переводе на язык программирования выражение $x^2 - 2 = 0$ следует понимать не как требование присвоить левой части равенства значение правой, а как утверждение о равенстве левой и правой частей, истинность которого еще необходимо проверить. Иначе говоря, с точки зрения языка программирования данная запись соответствует операции сравнения выражений слева и справа от знака равенства. Результатом этого может быть одно из двух: ИСТИНА (если левая и правая части оказались равными) или ЛОЖЬ (в противном случае). Таким образом, выражение $x^2 - 2 = 0$ следует трактовать как "проверить, выполняется ли равенство $x^2 - 2 = 0$". Разумеется, такая проверка предполагает подстановку в его выражение конкретного значения вместо символа "x".

Зануда. Если я правильно Вас понял, то знак равенства в выражениях языка может играть две совершенно различных роли. Во-первых, в простейших выражениях вида $x = 5$ он обозначает повеление присвоить переменной x некоторое значение (в данном случае 5). Во-вторых, в выражениях вида $x^2 - 2 = 0$ тот же знак равенства следует трактовать как операцию сравнения того, что находится слева от него, с тем, что расположено справа. Лишь по самой форме выражения мы можем определить, каков смысл употребления символа "=". Однако при изучении математики (вспомните школу!) с этим не возникало особых проблем.

Профессор. Ваша проницательность, уважаемый Зануда, выше всяких похвал. Именно потому, что один и тот же знак равенства может выполнять в математических выражениях различные роли, в языках программирования для каждой из этих ситуаций обычно предусматривают свой знак. Так например, для присваивания переменной некоторого значения обычно используют символ "=", а для сравнения значений двух выражений на предмет их равенства — двойной символ равенства "==". Например, выражение $y = x^2 - 1$ в программе следует понимать как команду присвоить переменной y значение выражения $x^2 - 1$, а выражение $y == x^2 - 1$ — как операцию сравнения значений y и $x^2 - 1$. Таким образом снимается проблема неоднозначной интерпретации операций, которые должен произвести компьютер. Замечу попутно, что в некоторых языках для присваивания употребляют символ ":=", а для сравнения — "=". Однако все это — особенности синтаксиса конкретных языков программирования, которые легко освоить с помощью справочников. Никогда синтаксис языка, каким бы вычурным он ни был, не создавал настоящих проблем для его изучения. Трудности возникают при словоупотребления и формировании предложений. Человек, изучающий не первый в своей жизни язык программирования, легко и непринужденно усваивает его скучные нотации и спецификацию потому, что уже имеет некоторый опыт составления осмысленных предложений на искусственных языках.

А теперь в качестве тестового примера рассмотрим выражение вида $x = x + 1$. С точки зрения математики данное уравнение не имеет решения, поскольку легко преобразуется к виду $0 = 1$, что, очевидно, никогда не выполняется (данное равенство ложно). А раз так, то оно нам и не интересно. Однако с точки зрения программирования подобная запись, означающая присвоение переменной x значения выражения $x + 1$, имеет смысл. Что должен сделать компьютер, выполняя данное выражение?

Простак. Компьютер должен, как мне кажется, к текущему, т. е. ранее заданному значению переменной x прибавить единицу и полученный результат присвоить этой же переменной x. Другими словами, мы можем в качестве нового значения переменной использовать результат каких-то преобразований ее предыдущего значения.

Профессор. Совершенно верно. Так что, выражение $x = x + 1$ в языке программирования представляет собой не требование (императив) решить уравнение относительно переменной x, а команду увеличить существующее ее значение на единицу.

Итак, в математике переменная, обозначаемая каким-нибудь именем из одного или нескольких символов, представляет (обозначает) собой некоторое значение из заранее оговоренного множества допустимых величин. В языках программирования понятие переменной имеет аналогичный смысл — это контейнер для хранения данных. Данные, которые могут храниться в переменной, называют значениями этой переменной. Они могут быть весьма разнородными: числами, текстовыми строками, датами, массивами и др.

Данные группируют в множества, называемые типами данных. Простейшие примеры типов данных — числа и произвольные последовательности символов (строки), встречаются и другие, например, дата, время и т. д. Переменные классифицируют в зависимости от типа данных, которые они могут хранить. Так например, если переменная предназначена для хранения чисел, то говорят, что она имеет числовой тип или, короче, что это числовая переменная.

Допустим, вы имеете три яблока и одно из них отдали мне. Сколько яблок у вас осталось?

Простак. Уважаемый профессор, вы, похоже, думаете, что мы законченные идиоты, не способные выполнить элементарные арифметические операции. Мы мало или даже ничего не знаем ни в программировании, ни в математике, но это еще не повод для унижения.

Профессор. Я и не сомневался, что сказку про Буратино вы читали. Во-первых, никогда не обижайтесь на своего учителя, поскольку показать ваше полное незнание или, в худшем случае, превратное знание — не главная его задача, а лишь способ акцентировать внимание на какой-то проблеме. Во-вторых, учитель всегда желает вам добра, даже тогда, когда кажется, что он вас недооценивает. Наконец, в-третьих, попытайтесь все же ответить на поставленный мной вопрос в контексте обсуждаемой темы о типах данных.

Простак. Ответ на ваш вопрос очень несложен: два яблока у меня останется.

Профессор. Большинство согласится с Вами, уважаемый Простак, но программисты смогут предложить и другие варианты ответа.

Зануда. В жизни, как только встречаемся с числами, мы обычно абстрагируемся от того, к чему они относятся, к яблокам, автомобилям или еще чему-то. Поскольку в данном случае речь идет об изъятии яблока, то мы считаем, что этой операции соответствует обыкновенное вычитание одного числа из другого. Таким образом, мы

абстрагируемся от яблок, оставляя в поле зрения только их количества, а передачу яблок другому лицу интерпретируем как арифметическую операцию вычитания или прибавления. Это настолько естественно, что и программист должен рассуждать аналогично.

Профессор. Браво, Зануда! Абстрагирование и интерпретацию данных, а также операций над ними в обычных и несложных житейских ситуациях наш мозг производит довольно быстро и в большинстве случаев правильно. Однако программист задает вопросы не только людям, а главным образом компьютерам, интеллектуальные способности которых пока значительно уступают человеческим. Поэтому он должен добиваться точных формулировок необходимых определений исходных данных, задач и алгоритмов их решения. Иначе он рискует остаться только на уровне благих пожеланий с нулевыми шансами что-либо сделать посредством компьютера.

В рассматриваемом нами случае программист мог выполнить все необходимые интерпретации условия задачи в уме, а машине поручить лишь рутинную операцию арифметического вычитания над числами 3 и 1, а именно вычислить значение выражения 3 – 1. Компьютер, очевидно, выдал бы в результате число 2, а программист сказал нам: "останется два яблока". Однако возможен и иной ход событий. Если в компьютер ввести данные в другой форме: "3 яблока" и "1 яблоко", то что получится в результате попытки вычислить выражение `"3 яблока" - "1 яблоко"`?

Простак. Я понял, что в рассмотренной ситуации делается попытка выполнить арифметическую операцию не над числами, а над символьными строками, пусть даже и содержащими числа. Это, возможно, поставит компьютер в тупик.

Зануда. Тупик будет более очевидным, если мы предложим компьютеру выполнить такую операцию: `"Саша" - "Маша"`. Данные, участвующие в этой операции, не содержат чисел, а потому у нас нет никакого разумного основания проинтерпретировать действие, обозначенное символом "минус", как арифметическое вычитание. Что должно получиться в итоге? Каким должно быть слово, являющееся результатом данной операции?

Профессор. Я не исключаю, что можно придумать некоторую особую интерпретацию этой операции. Например, пусть это будет буквосочетание, которое в качестве фрагмента есть в первом, но отсутствует во втором слове. Однако такое определение слишком специфично или, иначе говоря, не общезначимо, а потому нам проще объявить данное и подобные выражения лишенными смысла и запретить их использовать в конструкциях языка программирования.

В жизни мы имеем дело с различными данными, относя их к разным типам по той простой причине, что для одного типа данных имеют смысл и допустимы одни операции, а для другого — другие. То, что можно делать с числами, интерпретируя операции над ними в математике, нельзя выполнять с последовательностями произвольных символов, сохраняя при этом интерпретацию результата в той же математике. Наоборот, мы можем склеивать две последовательности символов или отсекать от них ненужные части, но результат подобных операций можно осмыслить,

например, в редакторской деятельности, но только не в математике. Поэтому последовательности произвольных символов (строки) выделяют в отдельный тип данных. Далее, дату можно представить в виде набора из трех чисел (число, месяц и год), которые обычно отделяют друг от друга в одной записи точками. Очевидно, что сложение двух дат едва ли можно понять как сложение двух чисел. Набор из нескольких чисел — более сложный объект, чем отдельное число. Однако прибавление к дате некоторого обычного числа вполне разумно интерпретировать как некую новую дату, следующую за исходной спустя заданное число дней. Вычисление этой даты, т. е. представление в виде набора из трех чисел не совсем тривиально (нужно знать, сколько дней в каждом месяце и году). Иначе говоря, операции над датами отличаются от простых арифметических операций над обычными числами, а потому даты также относят к особому типу данных. В языках программирования существуют так называемые встроенные типы данных. Это общезначимые, наиболее часто употребляемые и достаточно отчетливо интерпретируемые всеми типы. Вместе с тем, современные языки программирования допускают задание программистом своих особенных типов данных. Например, вы можете определить тип данных НЕДЕЛЯ, который содержит 7 элементов, представленных целыми числами (например, 1, 2,…,7) или названиями дней (например, "понедельник", "вторник",…, "воскресенье").

Во многих языках (например, C и Pascal) переменные и их тип должны быть объявлены прежде, чем они будут использованы в программе. Если при этом переменная изначально была объявлена, например, как числовая, то уже нигде в программе ей нельзя присвоить значение какого-нибудь другого типа. Говорят, что это языки с сильным контролем типов.

Однако есть языки со слабым контролем типов (их еще называют языками со свободными или динамическими типами), типичные примеры — JavaScript и PHP. В них тип переменной не фиксируется раз и навсегда в пределах программы, одна и та же переменная в различных фрагментах программы может принимать значения то одного, то другого типа по усмотрению программиста.

Простак. Хотелось бы все это рассмотреть на примерах.

Профессор. Пока мы не будем углубляться в детали реализации идеи переменных в конкретных языках. Замечу также, что мы не затронули еще одну важную тему, связанную с переменными, а именно области их видимости. Дело в том, что переменные могут быть глобальными и локальными. Глобальные доступны (т. е. видны) из любого места программы, локальные — только в том блоке программного кода (фрагменте программы), где они были созданы. Типичный пример блока кода — функция. Понятие функции мы рассмотрим позже, а пока лишь отметим, что она может содержать в своем коде переменные, недоступные (невидимые) из внешней по отношению к ней части программы. Мы изучим данную тему далее применительно к двум языкам — JavaScript и PHP.

2.2. Массивы данных

Профессор. Ранее мы рассмотрели в общих чертах, что такое переменные и типы данных. Среди последних выделяют простые или примитивные типы, такие как числа, строки символов, логические значения (ИСТИНА и ЛОЖЬ). Они потому и называются примитивными, что интуитивно понятны, однако наш разум требует чего-то большего. А именно, он без особых усилий обнаруживает, что могут быть и более сложно устроенные данные. Такие типы называют составными. Например, даты, представляющие собой наборы из трех целых чисел, очевидно сложнее, чем просто одиночные числа. Особенно важны с практической точки зрения массивы данных, т. е. наборы не двух-трех, а произвольного количества данных. Наиболее общее определение массива — упорядоченная последовательность данных.

Простак. Иначе говоря, массив состоит из элементов, следующих друг за другом в определенном порядке. К настоящему времени мы усвоили кое-что о примитивных типах данных — числах, строках и логических значениях. А какого типа могут быть элементы, входящие в последовательность, именуемую массивом? Я легко представляю себе последовательность чисел, символьных строк и даже значений типа ИСТИНА и ЛОЖЬ. Но если допустимы данные и других, более сложных типов, то можно сотворить что-то грандиозное.

Профессор. В некоторых языках (например, в C и Pascal) массив определяется как упорядоченная последовательность данных одного и того же примитивного типа, например, массив целых чисел, массив символьных строк и т. п.

Простак. А жаль. Если бы в массив можно было "загнать" данные сложных (составных, как Вы их называете) типов, то появилась бы возможность реализовать двумерные или даже *n*-мерные таблицы.

Зануда. Мне не совсем понятно, на что намекает Простак. Меня больше интересует само понятие массива, а потому я хотел бы его уточнить. Вы говорите, профессор, что массив это упорядоченная последовательность данных. А если изменить исходный порядок сортировки данных? Станут ли два массива, содержащие одни и те же данные, но расположенные по-разному, эквивалентными, или, лучше сказать, одинаковыми с точки зрения интерпретатора, т. е. того, кто будет выполнять наши программные коды?

Профессор. Изменение первоначального порядка следования данных (элементов массива) приводит к другому массиву. Так что, массив это не просто "куча данных", их порядок в определении массива имеет принципиальное значение. Определяя массив как последовательность данных, мы закрепляем память о нем, присваивая его некоторой переменной, т. е. назначая ему имя подобно тому, как мы это делаем с данными примитивных типов. Иначе говоря, мы назначаем по своему достаточно произвольному выбору имя переменной и присваиваем ей значение, являющееся массивом.

Массив, подобно обычной переменной, имеет имя, так что одним именем мы можем обозначить не одно, а сразу несколько простых значений. Другими словами,

переменной типа массив можно присвоить набор примитивных значений, упорядоченных произвольным образом.

Простак. Массив — это последовательность данных, причем упорядоченная. На практике обычно требуется не просто обратиться к массиву в целом, а узнать конкретное значение какого-то определенного элемента этого массива. Чтобы получить доступ к значению обычной переменной (т. е. переменной примитивного типа данных), достаточно указать в программе имя этой переменной. А как получить нужное значение элемента массива?

Зануда. Поскольку массив задается не просто своими элементами, но и порядком их следования, то, наверное, можно как-то указать номер нужного элемента. Насколько я правильно понимаю, любой порядок всегда связан с возможностью нумерации целыми числами.

Профессор. Совершенно верно. Обычно элементы массива нумеруются целыми числами, начиная с нуля, а не с единицы (первый элемент массива имеет номер 0, а не 1). Номер элемента массива еще называют индексом. Так, если a — массив из n элементов, а нам требуется i-й элемент, то пишут $a[i]$. При этом если вам нужен самый первый элемент массива, то следует обратиться к $a[0]$, а не к $a[1]$. В частности, значение последнего элемента есть $a[n-1]$, а не $a[n]$. Впрочем, должен вас предупредить, что данное правило не повсеместно. Если я не ошибаюсь, то в языке VBScript это не так.

Простак. Можно сказать, что различные элементы массива обозначаются одним и тем же именем, но с различными индексами. При этом имена элемента и массива, которому он принадлежит, должны совпадать, а индекс — быть целым числом в диапазоне между нулем и значением, на единицу меньшим количества всех элементов.

Профессор. Правильно. Добавлю только, что во многих языках обращение к элементам массива по индексу — не единственно возможный способ, но об этом позже.

Зануда. Наглядно массив можно представить себе в виде таблицы из одного столбца, в ячейках которой находятся значения элементов.

Профессор. Это хорошая интерпретация массива.

Зануда. Мне не дают покоя намеки Простака на дальнейшее развитие понятия массива. Не знаю, о чем он думал, но меня сейчас интересует, а как создать таблицу с несколькими столбцами?

Простак. Позвольте, профессор, я сам попытаюсь ответить Зануде. Думаю, что это можно сделать, задав массив, элементы которого сами являются массивами. Последние будут определять строки или ряды прямоугольной таблицы, если, конечно, все они состоят из одинакового количества элементов. Тогда, чтобы получить данные из конкретной ячейки таблицы, потребуются два индекса: первый будет указывать на строку, а второй — на столбец (ячейка таблицы, как известно, — пересечение строки и столбца).

Зануда. Однако в таком случае исходный массив будет содержать не примитивные данные, а составные. Вы же, профессор, говорили раньше, что массив должен содержать данные одного и того же примитивного типа.

Профессор. Я говорил лишь, что это требуется в некоторых, но не во всех языках. Так например, в JavaScript, PHP и ряде других языков массив может состоять из разнотипных и не обязательно примитивных данных.

Зануда. Мы много говорили о массивах как о некоторых существующих объектах. Но объект, с которым предстоит иметь дело в программе, следует, прежде всего, как-то задать. А как определить массив?

Профессор. Хорошо, уважаемый Зануда, что Вы вспомнили об этом. В различных языках это делается по-разному, а нередко и несколькими способами. Например, в JavaScript массив с именем, скажем, `myarray` можно задать с помощью такого выражения:

```
myarray = [150, 200, "Вася", "Петя", "Маша"]
```

В квадратных скобках перечисляют значения элементов массива, разделенные запятыми. Как я уже упоминал, такой синтаксис не универсален. В различных языках задание массива оформляется по-разному. Рассматривая приведенное выражение как задание массива в JavaScript, мы можем отметить, что здесь значением элемента, например, `myarray[1]` является число 200, значением `myarray[2]` — символьная строка "Вася".

Зануда. Я вижу, что понятие массива в языках JavaScript и PHP было усовершенствовано так, чтобы элементы массива могли быть разнотипными и даже составных типов. С точки зрения удобства практических применений это здорово. Однако исходное достаточно простое понятие массива оказалось размытым. Его следовало бы скорректировать хотя бы так: "массив — это упорядоченная последовательность данных". При этом мы не акцентируем внимание на типах данных, а следовательно допускаем любые типы, предусмотренные в языке.

Профессор. Понятие массива в исходной и весьма ограниченной формулировке было первоначально реализовано в языках программирования как обращение к одноименным данным, расположенным рядом друг с другом в памяти компьютера. Это позволило обеспечить быстрый доступ к любому элементу из сколь угодно большого множества. Изначально здесь аппаратно-программное представление довлело над общезначимым или пользовательским. Понятно, что если заполняющие неразрывную область физической памяти данные однотипны, то поддержать технически (и средствами языка) такую конструкцию проще, чем в случае разнотипных данных.

Однако даже фундаментальные понятия подвержены изменениям с течением времени. Чтобы расширить первичное понятие массива как набора однотипных данных, в языке C была введена конструкция, называемая структурой. Структура — это набор элементов (называемых еще полями), которые предназначены для хранения данных различных типов. В языках JavaScript и PHP структур нет, но существуют массивы разнотипных данных. Это расширенное понятие массива могло быть реализовано в указанных языках посредством структур языка C. Мы не будем

сейчас рассматривать, что такое структуры. Программист может и не знать, что и каким именно образом реализовано. Например, он может понимать массивы в JavaScript и PHP как структуры в языке C, просто называя их массивами. В действительности же расширенное понятие массива в JavaScript и PHP было реализовано посредством не структур, а объектов, т. е. еще более обобщенных понятийных и технологических образований. Объекты — обобщение структур в том смысле, что они могут хранить, как контейнеры, не только данные различных типов, но и функции (называемые методами), т. е. некие элементы, обладающие способностями выполнять некоторые действия. В частности, когда объект содержит только переменные для хранения данных, но не имеет собственных функций (методов), он практически ничем не отличается от структуры или, если угодно, от массива с произвольными типами данных элементов. Но об объектах мы поговорим отдельно. Прежде следует выяснить, что такое функция.

2.3. Функции

Профессор. Программа представляет собой последовательность выражений языка. Нередко случается, что какая-то часть программы (блок кода) неоднократно повторяется. Чтобы устранить подобного рода избыточность кода, используют понятие функции. Функция — это поименованный блок кода, который вызывается в нужных местах программы по имени. Другими словами, функция представляет собой подпрограмму, которую можно вызвать из основной программы, причем неоднократно. Повторяющийся (да и не только) блок программного кода обычно обозначают некоторым уникальным именем, чтобы потом при необходимости обратиться к нему по этому имени. Как видно, это простая и естественная идея, направленная на облегчение реализации сложных проектов, состоящих из более простых программ.

Подпрограмма или, другими словами, функция должна быть связана (интегрирована) с основной программой, так сказать, со своим внешним окружением. С целью обеспечения взаимодействия с остальной частью программы для функции можно предусмотреть так называемые вход и выход. Вход в функцию — это передача ей аргументов — данных, полученных во внешней части программы. Получив данные из своего внешнего окружения (внешней программы), функция должна их как-то обработать: выполнить некоторые действия, вычислить какое-то значение. Выход из функции — значение, вычисленное блоком кода данной функции, и передаваемое во внешнюю часть программы. Входные данные называют параметрами, а выходные — возвращаемым значением. Впрочем, функция может и не принимать никаких параметров, а также ничего не возвращать. Что принимает в качестве параметров, и что возвращает функция в результате своей работы, определяет программист, т. е. автор-разработчик программного кода.

Функция, создаваемая разработчиком, должна иметь определение, а ее применение в программе называют вызовом функции. Например, в JavaScript и PHP (а также в большинстве других языков) такое определение начинается с ключевого слова `function`, за которым следуют имя функции, пара круглых скобок, внутри которых

можно указать список параметров, а затем блок кода, заключенный в фигурные скобки:

```
function имя_функции(список_параметров){
      блок кода
}
```

Замечу попутно, что блок кода в фигурных скобках называют еще телом функции. В нем записывают выражения языка, которые функция должна выполнить, если будет вызвана из основной программы. Таким образом, само по себе определение функции в программе еще ничего не выполняет. Оно только указывает интерпретатору, что должно быть выполнено при обращении к данной функции.

Для вызова функции в том или ином месте программы указывают следующее выражение:

```
имя_функции(список_параметров)
```

Круглые скобки после имени функции записывают независимо от того, предусмотрены для нее входные параметры, или нет.

В программе вы можете один раз задать определение функции, а затем, по мере необходимости, вызывать ее столько раз, сколько требуется. Таким образом, вам не придется многократно воспроизводить в программе код тела функции.

Простак. Верно ли я Вас понял, что основная программа может содержать как определения, так и вызовы функций?

Профессор. Да, это так. В программе, содержащей вызовы функций, должны быть доступны их определения. Простейший, но не единственно возможный способ обеспечить это, заключается в том, чтобы заключить определения функций в тот же программный код, в котором они где-то вызываются. Опытные программисты обычно размещают определения функций в одном месте, например, в конце или, наоборот, в начале кода программы.

Простак. Мне понятно, что в некоторых ситуациях может потребоваться передать какой-то функции данные из основной программы. Но зачем функция должна что-то возвращать? Ведь она может выполнить предусмотренные действия, например, отправить данные по электронной почте, открыть или закрыть окно, отобразить картинку и т. п., а внешней программе не потребуется от нее какое-либо значение.

Профессор. Если вызывающей программе ничего не требуется получить от вызванной функции, то программист может определить эту функцию как ничего не возвращающую. Это разрешается в JavaScript и PHP, но может быть недопустимо в других языках. Если даже практически не важно, что именно возвращает функция, все равно желательно определить для нее некоторое возвращаемое значение. В таких случаях, часто указывают, например, пустое значение, обозначаемое как null. Возможны и другие варианты. Кроме того, функции, возвращающие какие-то значения, можно использовать в выражениях с операторами (например, арифметическими), а также в качестве параметров других функций. Если функция ничего не возвращает, то ее применение в выражениях с операторами обычно чревато сообщениями об ошибках.

Чтобы функция что-то возвращала, необходимо записать в ее теле специальный оператор возврата:

```
return возвращаемое_значение;
```

Приведу пример (листинг 2.1).

Листинг 2.1

```
Оклад = 10000;
Процент = 15;
Выплата = Оклад + Премия(Оклад, Процент);
function Премия(Оклад, Процент){
return Оклад*Процент/100;
}
```

В листинге 2.1 переменным `Оклад` и `Процент` сначала присваивают некоторые числовые значения. Затем вычисляют выражение, определяющее объем выплаты некоторому сотруднику с учетом его оклада и премии. Премия рассчитывается с помощью функции `Премия(Оклад, Процент)`, принимающей два параметра: оклад и процент от оклада; данная функция возвращает величину премии (символом `*` обозначен оператор арифметического умножения). Возвращаемое значение прибавляется к значению переменной `Оклад`, а полученный результат присваивается переменной `Выплата`. Определение функции `Премия(Оклад, Процент)` размещено в конце текста программы.

Зануда. А на каком, собственно, языке написан данный программный код?

Профессор. На вымышленном. Я хотел сказать, что рассмотренный пример иллюстрирует принцип применения функции, а потому не важно, какой именно язык выбрать. Главное, что все изложенное ранее должно позволить вам понять данный код. Тем не менее, признаюсь, что приведенный мной пример написан и на языке JavaScript. Напишите в обычном текстовом редакторе, например, Блокноте Windows следующие строки (листинг 2.2).

Листинг 2.2

```
<HTML>
<SCRIPT>
Оклад = 10000;
Процент = 15;
Выплата = Оклад + Премия(Оклад, Процент);
function Премия(Оклад, Процент){
return Оклад*Процент/100;
}
alert("Выплата = "+Выплата);
</SCRIPT>
</HTML>
```

Это HTML-код с внедренным в него сценарием на JavaScript. Код сценария заключен между тегами <SCRIPT> и </SCRIPT> и отличается от рассмотренного ранее только встроенной функцией alert() для вывода сообщений в диалоговом окне. Сохраните данный код в файле с расширением htm или html, а затем откройте его в Web-браузере, например, Microsoft Internet Explorer или Mozilla Firefox. В результате появится диалоговое окно с сообщением "Выплата = 11500".

Зануда. Однако я не вижу особой целесообразности применять здесь функцию, поскольку нет ее повторного использования. Было бы проще написать более короткий код (листинг 2.3).

Листинг 2.3

```
<HTML>
<SCRIPT>
Оклад = 10000;
Процент = 15;
Выплата = Оклад + Оклад*Процент/100;
alert("Выплата = "+Выплата);
</SCRIPT>
</HTML>
```

Профессор. В данном случае вы правы. Однако мой пример — лишь иллюстрация работы с функцией, без учета целесообразности ее конкретного применения.

Появление в 1960-х годах в языках (например, ALGOL и FORTRAN) функций позволило писать программы, более ясные в структурном отношении и компактные по объему кода. В идеале основная программа могла состоять из одних только вызовов функций, перемежаемых, возможно, операторами управления (условными переходами и/или циклами). Разрабатывать и отлаживать такие программы стало и быстрее и легче. Программист при желании мог писать программы, широко используя вызовы функций, тела которых еще не написаны. Чтобы такая программа как-то работала без зависаний и выдачи системных сообщений об ошибках, вместо настоящего кода функции можно было временно поставить так называемые заглушки, например, выводы специальных сообщений программиста, например, "работает функция myfunc". По появлению подобных сообщений можно судить, что вызов функции произошел, и при этом в главной программе не возникло коллизий.

Создав основную программу в общем виде, можно приступить к доработке ее деталей — написанию требуемых кодов, составляющих тела настоящих функций, которые до сих пор были замещены примитивными заглушками. Это так называемый способ (или стиль) программирования "сверху вниз", при котором создаваемая программа постепенно обретает требуемые функции. Усовершенствование такой программы впоследствии сводится главным образом к модификации кодов уже существующих функций и может быть выполнено другими людьми, а не только автором программы. При этом, разумеется, необходимо сохранить прежний

интерфейс (вход-выход или связи с внешним окружением), чтобы не пришлось перестраивать внешнюю программу, содержащую вызовы модифицируемой функции.

Даже если какой-то блок кода предполагается задействовать в программе всего лишь один раз, все равно следует подумать, а не оформить ли его в виде функции, чтобы сделать основной текст программы более понятным и к тому же обеспечить возможность повторного использования этого блока в перспективе.

Зануда. Я заметил еще одно странное обстоятельство: функции `alert()` в качестве параметра передается выражение в виде суммы разнотипных данных (символьного и числового значения):

```
"Выплата = "+Выплата
```

Здесь `"Выплата = "` — символьная строка, а `Выплата` — переменная числового типа.

Однако Вы, уважаемый профессор, ранее говорили, что типы данных для того и придуманы, чтобы выполнять операции над однотипными данными.

Простак. Тем не менее, сообщений об ошибках не было, а результат оказался правильным!

Профессор. Рассмотренный сценарий написан на JavaScript, т. е. на языке со слабым контролем типов данных. Если в выражении этого языка встречается операция над данными различных типов, то происходит такое автоматическое преобразование этих данных, при котором может быть получен более или менее осмысленный результат. Так например, если требуется к символьной строке прибавить число, то интерпретатор JavaScript сначала преобразует его в соответствующую строку символов, которую затем "приклеит" к первой строке. В результате получится символьная строка. Обратите внимание, что число, записанное как последовательность цифр (возможно, с разделительной десятичной точкой и знаком "+" или "–" перед числом), с точки зрения типа данных отличается от того же числа, заключенного в кавычки. В последнем случае это будет строка, содержащая число, а не собственно число. В языках со слабым контролем типов операции над разнотипными данными выполняются с учетом некоторых правил автоматического преобразования типов (так называемые правила приведения типов), которые по умолчанию разные для различных языков. Поэтому многие программисты, чтобы не запутаться и все держать под своим полным контролем, выполняют предварительное приведение данных к нужному типу с помощью специальных встроенных функций, а не полагаются на автоматические преобразования. Однако хорошее знание особенностей языка позволяет создавать более короткие и элегантные программы.

Простак. Вы использовали в коде, вставленном в HTML-документ, функцию `alert()` для вывода на экран диалогового окна с некоторым сообщением. А откуда она взялась? В сценарии же нет ее определения.

Профессор. Если быть точным, то `alert()` — не функция языка JavaScript, а метод объекта `window`, который принадлежит объектной модели браузера. Методами называют внутренние функции объектов. Иначе говоря, этот метод, как и множество других объектов браузера и загруженного в него документа, составляют внешнее

окружение сценария на JavaScript. К объектам этого окружения можно обратиться из сценария, т. е. прочитать и даже изменить значения их свойств. Тем самым обеспечивается возможность управлять внешним видом и информационным содержанием Web-страницы.

Простак. Но чтобы реализовать все это, необходимо знать устройство или, лучше сказать, объектную модель документа и браузера.

Профессор. Разумеется, модель того, с чем требуется иметь дело посредством языка программирования, необходимо знать. Но это особая тема. Сейчас же мы рассматриваем языки программирования, причем с высоты "птичьего полета" и до поры не будем слишком приближаться к "земле".

Зануда. Осмелюсь напомнить, уважаемый профессор, что ранее вы нам что-то сказали о видимости переменных. В частности, Вы заметили, что в коде функции (ее теле) можно задать локальные переменные, которые недоступны во внешней части программы.

Профессор. Да, в теле функции можно определить переменные, невидимые в ее внешнем окружении. Даже если имена внешних и внутренних (локальных) переменных совпадают, то их значения могут отличаться. Однако подробное изучение этой темы лучше провести на примере конкретных языков, что мы и сделаем немного позднее.

2.4. Классы и объекты

Профессор. При моделировании какой-то части реального мира мы вводим в обиход некоторые сущности (предметы) и отношения (связи) между ними. Предмет или объект, каким бы он ни был в действительности или в нашем воображении, в абстрактном виде можно представить себе как нечто, обладающее определенными свойствами. Об объектах окружающего нас мира, а также нашей мысли мы судим по их свойствам, т. е. по тому, что можем воспринимать нашими органами чувств. Иначе говоря, объект внешнего или нашего внутреннего мира представляется нам не совсем таким, каким он выступает в действительности, а лишь посредством тех свойств, которые мы способны воспринять. Здесь мы, похоже, стучимся в дверь философии. Однако пора приземлиться, коль скоро мы занимаемся прикладной наукой. Попробуйте, друзья, для начала описать такой объект, как автомобиль.

Зануда. Я сразу представляю себе с помощью зрительной памяти один или даже несколько автомобилей различных марок и окрасок. Несмотря на то, что они различаются между собой внешне и внутренне, существует нечто общее между ними, но именно это общее я и затрудняюсь сформулировать во всей исчерпывающей данное понятие полноте.

Простак. Тебе, Зануда, всегда удается сказать что-то такое, возразить чему очень трудно. Тем не менее, я тоже способен к разглагольствованиям отвлеченного порядка. В контексте нашего разговора я способен признать, что между различными автомобилями есть что-то общее, позволяющее отнести их к одному и тому же классу, а именно классу автомобилей. Осмелюсь заметить, что это по-видимому

наличие колес и двигателя внутреннего сгорания. Не спешите поймать меня на слове и "схватить за язык", уважаемые Зануда и профессор. Разумеется, это не все и, возможно, не самые главные особенности, с помощью которых мы можем выделить автомобиль из окружающего мира. Тем не менее, по данным признакам мы все-таки отличим автомобиль от паровоза, корабля или обезьяны, например, а это уже кое-что.

Зануда. Выделить объект или, как говорят, сущность из окружающего мира, — трудная задача, тем более при первом рассмотрении. Но мы должны это делать, чтобы как-то справиться со сложностью реального мира при попытке его смоделировать, а значит — познать.

Профессор. Вот! Уважаемый Зануда сформулировал самое главное, к чему стремятся не только программисты, создающие свои опусы на искусственных языках для компьютеров, но и математики, пишущие свои пьесы на нетленном языке мысли, хотя и снабженном специфической символикой. Все они, философы, математики, программисты, поэты и обычные люди, говорящие на языке, ставшем им доступным и родным просто от рождения, трансформируют окружающий нас многогранный, с трудом постигаемый и не объяснимый в полной мере, мир в свои внутренние модели. Зачем? А понять внешний мир иначе как его отражение во внутреннем невозможно. Впрочем, я так же как и вы, не очень-то искусный философ, а потому призываю вас сейчас вернуться к весьма практической теме нашего разговора.

Итак, объект — весьма абстрактная категория нашего сознания и языка как средства сделать эту мысль доступной не только нашему внутреннему взору, но и соседям. Мы должны как-то "приземлить" это довольно общее понятие, чтобы оно стало податливой для моделирования в довольно специфических и весьма "аскетических" искусственных языках, а значит и в программировании.

Объекты отличаются друг от друга составом и/или значениями (параметрами) свойств. Объект — это своего рода контейнер, содержащий информацию о себе самом. Какова эта информация? Очевидно, что это сведения о свойствах объекта. Но каким образом сохранить эту информацию для последующего использования?

Простак. Чтобы сохранить какую-то информацию на будущее, мы ее запоминаем. В естественных условиях это как-то делает мозг, способный в дальнейшем извлечь на поверхность нашей памяти (лучше сказать, — сознания) с помощью заранее заготовленной ссылки — имени интересующей нас информационной области. Когда же мы создаем компьютерную программу, запоминание происходит путем присвоения данных, содержащих интересующую нас информацию, некоторой переменной. Конкретно, мы просто назначаем данным символическое имя, через которое впоследствии можно получить доступ к интересующим нас данным.

Профессор. Я бы не сказал лучше, чем Вы, уважаемый Простак.

Зануда. Я удивлен, как тебе, Простак, удается сходу все это понять и даже выразить достаточно членораздельно?

Простак. А я просто стараюсь относиться к излагаемому нашим профессором без предубеждений, не противопоставляя ему без особой надобности свой предшествующий опыт. Мне, в конце концов, нужен именно его опыт, а не подтверждение ценности своего собственного.

Зануда. Я привык, что ты обычно говоришь в безапелляционной манере. Отчего в твоих словах теперь проглядывает осмотрительность, которой раньше недоставало?

Простак. Правда, я не склонен излишне мудрствовать. Я живу, пока вижу цель и путь к ее достижению. Вместе с тем, я не могу быть счастлив только тем, чтобы барахтаться в грязи дороги, пусть и ведущей к сияющей чистотой цели.

Зануда. Чтобы не "барахтаться в грязи", как ты говоришь, надо взлететь или хотя бы подпрыгнуть над строящейся дорогой. А подпрыгнув, что ты оттуда сможешь увидеть? Ту же непроторенную грязь, пусть даже и с высоты белоснежных вершин?

Простак. Ты, Зануда, просто провоцируешь меня на то, чтобы я спустился с небес, где ты сам давно обосновался. Похоже, ты хочешь просто вытолкнуть меня оттуда, как кукушонок. Неужели из-за меня тебе там стало нестерпимо тесно?

Зануда. Не сердись, дорогой Простак. Мне там и в самом деле хорошо летать, но холодно и одиноко, а потому я искренне рад твоему беспокойному соседству. Однако наше сближение грозит мне потерей оппонента — того, кто не даст моим мыслям протухнуть. Поэтому, любя по большому счету, я подспудно стараюсь оттолкнуть тебя. Так что, прости.

Профессор. Ребята, не увлекайтесь выяснением личных отношений. Истина, к которой вы стремитесь, не обязательно благотворна с точки зрения ваших субъективных представлений о настоящем и будущем. Незнание истины лишь откладывает на будущее возможные очарования (и разочарования), создавая только иллюзию предвкушения счастья (или несчастья) в настоящем. Но пренебрежение уже доступной истиной, а также отказ от ее поиска однозначно чреваты неизбывными горестями впоследствии.

Мы с вами слишком отвлеклись от главной темы обсуждения. Напомню, что речь идет об объектах, т. е. таких программных конструкциях, которые характеризуются или, лучше сказать, предстают перед нами посредством своих свойств. Мы должны признать, что наше знание об объекте в целом заключается в знании его отдельных свойств. Такой подход к пониманию целого присущ и некоторым разделам математики. В частности, в аксиоматических теориях объект изучения задается набором аксиом, которые постулируют, какими свойствами он обладает. Так что объект в математике предстает перед нами как нечто (назовите и нарисуйте в своем воображении его как угодно), обладающее определенными свойствами. Далее математическая теория развивается, обрастая теоремами, чтобы сообщить нам информацию об объекте изучения, которая логически следует из первоначальных аксиом, т. е. утверждений, принимаемых без доказательства в качестве самоочевидных или достаточно обоснованных практическими соображениями. По мере развития теории об объекте мы получаем более или менее отчетливое представление о том, что есть собственно объект. Примерно так обстоят дела и при математическом моделировании объектов.

В программировании применяется довольно абстрактное и "технологичное" понятие об объекте как о контейнере, содержащем внутренние переменные (свойства), а также внутренние функции (методы). Внутренние переменные и функции можно называть свойствами объекта. Свойства-переменные предназначены для хранения

данных, а свойства-методы — для выполнения некоторых действий. Особенность и тех, и других в том, что в программе они играют не самостоятельную роль, а относятся к тому объекту, для которого были определены и с которым тесно связаны. Представьте себе какое-нибудь бытовое устройство, например, микроволновую печь или телевизор. С точки зрения пользователя его можно представить абстрактно в виде "черного ящика". Ящик называется "черным" потому, что его внутренности непосредственно недоступны ни органам чувств, ни уму. По крайней мере, мы, как пользователи, не хотим разбираться в премудростях его внутреннего устройства. Вместе с тем, к этому "ящику" подключены датчики или измерительные приборы (вольтметр, амперметр, манометр, термометр, спидометр и т. п.), показания которых мы можем наблюдать. Кроме того, имеются органы управления (кнопки, переключатели), с помощью которых мы можем заставить наше устройство выполнить ту или иную функцию. Нажимая на кнопку, мы инициализируем соответствующую ей функцию, т. е. заставляем наше устройство выполнить некоторое действие. При этом показания каких-то датчиков, возможно, изменятся. Иначе говоря, индикаторы могут реагировать на воздействия органов управления. С точки зрения программирования датчики это прообразы свойств-переменных, а органы управления — методов.

Простак. Однако я хотел бы заметить, что бытовые приборы с их индикаторами (датчиками) и кнопками (управляющими элементами) обычно снабжают инструкциями или описаниями того, как они работают и как правильно их использовать по назначению. Я обращаю внимание на то, что знание о свойствах-переменных и свойствах-методах это еще не все знание об объекте. По меньшей мере еще требуется информация о способах применения.

Профессор. Верно, описание объекта в виде инструкций и руководств для пользователя представляет не что иное как модель его функционирования, т. е. взаимосвязь показаний датчиков и управляющих воздействий. Объект в программе существует в виде набора свойств, но чтобы его использовать, необходимо знать его модель поведения и программный интерфейс. Модель поведения объекта обычно задается в его описании от производителя.

Простак. Программный интерфейс? А что это такое?

Профессор. Это описание (спецификация) взаимодействия какой-нибудь более или менее автономной программной единицы (например, объекта) с внешней программой. Он устанавливает правила работы с значениями свойств-переменных и методами. Другими словами, программный интерфейс определяет инициализацию объекта и способы взаимодействия с ним в вашей программе.

Введение в программирование понятия объекта обусловлено главным образом сложностью современных программ. Типичный пример подобных программ — игры, такие как "стратегии", "симуляторы" и "стрелялки". В них много разнообразных персонажей и объектов с нетривиальным поведением. Как хорошо известно, с проблемой очень большой сложности обычно справляются посредством классификации. Так, миллиарды представителей животного мира разделяют на классы, а последние — на подклассы (отряды, семейства, роды, виды и т. п.).

Представьте себе какой-нибудь автомобиль и попробуйте описать его, указав присущие ему свойства.

Простак. У автомобиля есть кузов, мотор, колеса и еще масса других свойств, которые можно упомянуть или нет в зависимости от конкретной задачи. Так, если нам необходимо отличить данный конкретный автомобиль от всех других, то понадобится очень много признаков.

Зануда. А чтобы отличить данный автомобиль от всех других существующих и мыслимых объектов, потребуется необозримо больший список признаков.

Профессор. Тем не менее, можно построить иерархическую систему классов интересующих нас объектов, постепенно переходя от классов к содержащимся в них подклассам и тем самым уточняя информацию об объекте. Например, класс всех автомобилей охарактеризуем наличием двигателя с приводом на колеса и системы рулевого управления. Из данного класса выделим два подкласса — легковых и грузовых автомобилей, отличая их друг от друга по особенностям устройства кузова и/или разрешенному максимальному весу. Подкласс легковых автомобилей разобьем еще на несколько подклассов следующего иерархического уровня по объему цилиндров двигателя, например. Подобным образом можно построить следующий уровень классификации. Количество уровней вложенности одних классов в другие зависит от требуемой степени детализации информации об объекте. Признаки или, другими словами, свойства классов также могут быть теми или иными в зависимости от наших возможностей их наблюдать и целей их последующего использования. Наконец, сама система классов может быть более или менее удачной. Важно то, что мы создаем сложное описание объекта постепенно, по частям. Причем на каждом этапе мы имеем дело с относительно простым и небольшим по объему объектом (классом). Говорят, что он наследует свойства своего надкласса, называемого еще родительским. Если какая-то система классов для описания всех интересующих нас объектов уже построена, то со временем некоторые из данных классов могут служить для описания других объектов. Например, может статься, что некоторые классы верхних иерархических уровней пригодятся для описания самолетов или иных транспортных средств. Следовательно, каждый подкласс кроме своих собственных свойств имеет еще и свойства класса, которому принадлежит. Такой подкласс содержит в собственном описании лишь специфические свойства. Свойства классов-родителей при этом в нем не описываются, а просто наследуются, т. е. заимствуются из описаний родительских классов.

Простак. А как создаются классы, неужели с помощью функций и переменных?

Профессор. Эра объектно-ориентированного программирования (ООП) началась в 1990-х годах. Современные языки, поддерживающие ООП (например, C++, Object Pascal, PHP и др.), позволяют создавать классы посредством специальных синтаксических средств, например, так:

```
class имя_класса {
      определение свойств-переменных
      определение свойств-методов
}
```

Как видите, класс это просто контейнер, содержащий описание объекта. С помощью этого описания можно создать один или несколько однотипных объектов или, другими словами, экземпляров класса. Например, пусть класс описывает такой объект как автомобиль конкретной модели (например, "Мерседес 600"), имеющий среди прочих свойств еще и цвет кузова. Тогда мы можем на основе общего описания автомобиля (т. е. класса) создать множество его экземпляров (объектов), различающихся конкретными значениями свойства *цвет_кузова* (черный, белый, красный и т. п.).

Зануда. Я совсем запутался в терминах *класс*, *экземпляр класса* и *объект*. Из Ваших слов я понял, что класс это описание какого-то объекта, а экземпляр класса — какая-то конкретизация этого класса, причем экземпляров одного и того же класса может быть сколь угодно много. Но чем тогда отличается подкласс от класса? Разве подкласс не конкретизирует свой родительский класс? А чем же отличается объект от экземпляра класса?

Профессор. Хорошо, давайте наведем порядок в терминологии. До сих пор мы употребляли слова "объект", "класс", "подкласс" и "экземпляр класса". Под объектом мы понимали некую реальную сущность, либо ее программную модель. Я полагал, что из контекста разговора достаточно ясно, о чем именно идет речь. Тем не менее, теперь я постараюсь более аккуратно обращаться с терминами.

Так вот, мы рассматриваем программное моделирование каких-то реальных сущностей, называемых объектами. Построить модель объекта означает составить ее описание. Такое описание может иметь иерархическую структуру, элементы которой называются классами. Иерархия классов выражает отношение родства или наследования свойств между классами. Не исключено также, что какой-то объект описывается единственным классом, который не связан ни с какими другими классами. Если объект описывается несколькими классами, связанными наследованием свойств, то каждый дочерний класс уточняет (дополняет) описание, представленное его родительским классом. Классы могут содержать как конкретные, так и неконкретные описания свойств. Например, в классе может быть объявлено свойство-переменная *цвет*, значение которой может быть присвоено или нет. Как бы то ни было, классы описывают некоторые объекты подобно справочникам, но сами по себе в программе еще не функционируют. Чтобы применить класс, его необходимо сначала, как говорят, инициализировать. Инициализация класса и есть создание его экземпляра. Она аналогична вызову обычной функции, заданной своим определением. Подобно тому, как вызываемой функции вы можете передать параметры, при создании экземпляра класса можно установить или переустановить значения некоторых или всех его свойств. Вы можете создать несколько экземпляров одного и того же класса, абсолютно одинаковых или отличающихся значениями своих свойств. Поэтому мы можем говорить, что экземпляр класса это некая конкретизация, но лучше сказать — конкретное воплощение класса в программе, из которой вы можете обращаться к свойствам и вызывать на исполнение методы экземпляра класса.

Зануда. Теперь, кажется, все встало на свои места.

Профессор. Однако я должен заметить, что у программистов все же имеется некий разнобой в терминологии. Так, нередко экземпляры класса называют просто объектами. Кроме того, существуют объектно-ориентированные языки (например, JavaScript), в которых классы не поддерживаются, а новые объекты создаются на основе уже существующих встроенных объектов. В таких языках термин "экземпляр класса" заменяют на "экземпляр объекта", а "класс" — на "объект".

Простак. А как создать экземпляр класса или, как еще говорят, объекта?

Профессор. Это делается с помощью специальной функции, называемой конструктором объекта. В различных языках задание конструктора и его вызов определяют по-разному, но основные принципы очень схожи. Более подробно эту тему мы рассмотрим при изучении конкретных языков, а сейчас лишь отмечу, что создаваемый экземпляр объекта можно присвоить обычной переменной, которая будет представлять его в программе. Данная переменная будет иметь особый тип данных object.

Простак. Хорошо, а тогда как обратиться к свойствам и методам экземпляров классов или объектов?

Профессор. Создав экземпляр объекта (класса), можно воспользоваться его свойствами — внутренними переменными и методами. Для этого необходимо указать не только требуемое свойство или метод, но и сам объект:

```
объект => свойство;
объект =>метод();
```

Здесь `объект` — это переменная, ссылающаяся на экземпляр объекта (класса) и играющая роль его имени в программе. Таким образом, свойство объекта обозначается не просто именем переменной, а составным именем, включающим в качестве префикса имя объекта. Аналогично образуется выражение для вызова метода. При этом два компонента обозначения разделяют специальным символом. В языках C++ и PHP, например, в качестве такого символа используют стрелку, а в JavaScript — точку:

```
объект.свойство;
объект.метод();
```

Таким образом, к свойствам-переменным и свойствам-методам мы обращаемся почти как к обычным переменным и функциям. Спецификой здесь является то, что перед именем свойства-переменной или вызовом свойства-метода необходимо указать объект, к которым они относятся.

Простак. Хорошо бы все изложенное Вами, профессор, рассмотреть на примерах решения конкретных задач. Иначе все эти тонкости трудно запомнить.

Профессор. Мы сделаем это, но немного позже, когда перейдем к изучению конкретных языков. А сейчас, завершая обсуждение объектов, я хотел бы отметить, что преимущества объектно-ориентированного стиля программирования очевидны в случае разработки достаточно крупных проектов. При решении задач относительно небольшой сложности, таких как создание Web-сайтов, обычно обходятся традиционными средствами языка и не оформляют программы в виде множества объектов. Программист должен применять те инструменты, которые он считает

удобными в конкретном случае, стараясь избегать неоправданных усложнений. Программы для Web-приложений, выполняемые как на стороне клиента (браузером), так и сервером, обычно невелики по объему и достаточно просты по структуре и логике работы, поэтому при их разработке можно отказаться от объектно-ориентированного подхода. Вместе с тем, сценарии для Web-сайтов пишутся на языках JavaScript и PHP, которые имеют встроенные объекты и предоставляют возможность работы с внешними объектами — объектами браузера и загруженного в него документа. Так, почти все конструкции JavaScript выполнены в виде объектов. Символьные строки, числа, массивы, функции — все это встроенные объекты со своими свойствами и методами. Поэтому даже если вы не собираетесь заводить в программе свои собственные объекты, все равно полезно знать, как они в принципе устроены и как с ними обращаться.

2.5. Операторы и выражения

Профессор. Программы состоят из операторов. Одно из важнейших действий, с которым мы уже встречались, — присваивание значения переменной, например, $x = 5$ или name = "Вася". Оператор присваивания обычно обозначают знаком равенства, а его результат состоит в том, что значение, указанное справа, помещается (записывается) в переменную, указанную слева.

Существуют и другие операторы, например, арифметические операторы сложения (+), вычитания (–) , умножения (*) и т. п. С их помощью составляют арифметические выражения для разнообразных вычислений, например, $x + 5$. Действие такого оператора заключается в том, что к значению переменной x прибавляется 5. Предполагается, что переменная x определена ранее и имеет некоторое числовое значение. Выражение $x + 5$ само получает значение, равное результату выполнения оператора. Говорят, что оператор (выражение с операторами) возвращает значение, полученное в результате его выполнения.

Простак. Кому возвращает?

Профессор. Программе, разумеется.

Зануда. А как мы можем воспользоваться полученным значением? Его ведь нужно сначала как-то запомнить.

Простак. Чтобы что-то запомнить, достаточно просто присвоить это какой-нибудь переменной.

Профессор. Совершенно верно. Мы можем выражение с операторами присвоить переменной: $y = x + 5$.

Зануда. Таким образом, Вы составили выражение с двумя операторами: присваивания и сложения. А в каком порядке они выполняются и какое значение будет возвращено данным выражением?

Профессор. Сначала к значению переменной x прибавляется число 5, а затем полученный результат помещается в переменную y. Это же значение и есть то, которое возвращается выражением целиком.

Теперь о порядке выполнения операторов. В общем случае различные операторы в одном и том же выражении выполняются в порядке, определяемом их приоритетами, а последние указываются в описании языка программирования. В частности, оператор присваивания имеет один из самых низших приоритетов.

Зануда. А есть ли операторы с одинаковыми приоритетами? Одинаковы ли приоритеты у операторов, скажем, сложения и вычитания?

Профессор. У многих операторов приоритеты одинаковы. В частности, сложение и вычитание имеют один и тот же приоритет, но меньший, чем у деления и умножения. В выражениях с одинаковыми по приоритету операторами последние вычисляются в порядке упоминания слева направо, если только с помощью круглых скобок не задан иной порядок. Например, значение выражения (2+3)*5 равно 25, а выражения 2+3*5 — 17. Впрочем, это вам должно быть известно еще со школы.

Операторы в программе обычно отделяются друг от друга точкой с запятой и выполняются друг за другом в порядке их следования слева направо и сверху вниз. Однако существует возможность изменить такой линейный порядок. Это обеспечивают так называемые операторы управления потоком вычислений, к которым относятся условные операторы и циклы.

Условный оператор позволяет реализовать структуру условного выражения:

```
если ..., то ..., иначе ...
```

В языках программирования этот оператор имеет следующий синтаксис:

```
if (условие)
 {код, который выполняется, если условие истинно}
else
 {код, который выполняется, если условие ложно}
```

В фигурных скобках располагается блок кода — одно или несколько выражений. Часть этой конструкции, определяемая ключевым словом `else` (иначе), необязательна. В этом случае остается только часть, определенная ключевым словом `if (если):`

```
if (условие){
 код, который выполняется, если условие истинно
}
```

Условие, указанное в круглых скобках за ключевым словом `if`, есть некоторое выражение логического типа, которое может принимать одно из двух значений: ИСТИНА или ЛОЖЬ. Обычно это выражение содержит оператор сравнения. Например,

```
if (x < 5)
 {y = 10; z = 3}
else
 {y = 0; z = -2}
```

Возможны и другие разновидности оператора условного перехода, но данный вариант базовый, присутствующий практически во всех языках.

Оператор цикла обеспечивает многократное выполнение блока программного кода до тех пор, пока не выполнится некоторое условие. Он имеет несколько вариантов, но базовыми являются операторы, начинающиеся с ключевого слова `for` (для)или `while` (до тех пор, пока…). При создании программ вполне можно обойтись каким-нибудь одним из них. Однако возникают ситуации, в которых один из операторов более удобен или естественен, чем другой. Синтаксис этих операторов и их применение мы подробнее рассмотрим позже, при изучении конкретных языков.

2.6. Специальные термины

Профессор. У программистов имеется профессиональный жаргон, с помощью которого они могут обмениваться мыслями более лаконично.

Простак. Да, я знаю, что вместо слова "программа" часто говорят "софт" или "прога". Кроме того, в Интернете сейчас довольно широко практикуется так называемый "олбанский язык".

Профессор. Вы привели примеры вульгарного жаргона, а "олбанский язык" вообще языком не является: это просто игра с изменением букв в словах так, чтобы последние звучали примерно так же, как и в нормальном языке. Например, "афтар" (автор), "привед" (привет), "йад" (яд) и т. п.

Мы сейчас познакомимся с той частью специальной терминологии, которая общепризнанна и широко применяется программистами при обсуждении языка и программ, как в устной, так и письменной речи.

Вот некоторые термины.

- Регистровая зависимость — зависимость выражений языка или их частей (например, имен переменных, функций и т. д.) от того, строчными или прописными буквами они написаны. Так, в регистрозависимых языках переменные myvar, MYVAR и myVar различны, а в регистронезависимых это одна и та же переменная. Например, язык JavaScript полностью регистрозависимый, а PHP — регистрозависимый, но не в полной мере.
- Идентификатор (identifier) — имя переменной, функции, объекта и др.; идентификатор не должен быть ключевым или зарезервированным словом (см. далее). Программист обычно выбирает идентификаторы по своему усмотрению, но в соответствии с определенными правилами. В большинстве случаев идентификаторы состоят из печатных символов (букв, цифр и символа подчеркивания), причем идентификатор не должен начинаться с цифры и содержать символы пробела. Примеры: `x`, `myvar`, `userName`, `user_name`, `var12`.
- Ключевое слово (keyword) — слово, являющееся частью языка. Ключевые слова недопустимы в качестве идентификаторов. Примеры: `function`, `if`, `while`, `var`.

- Зарезервированное слово (reserved word) — слово, которое не рекомендуется использовать в качестве идентификатора. Зарезервированные слова могут оказаться ключевыми в более поздних версиях языка.
- Оператор (statement) — команда, предписывающая компьютеру выполнить некоторое действие; операторы обычно приводят к изменению состояния окружения (например, переменной или определения). Примеры: `x = x + 5`, `function myfunc(x,y){return x*y}`.
- Литерал (literal) — значение, содержащееся непосредственно в тексте программы. Примеры: `5`, `"Привет всем!"`, `[3,7,9,25]`, `false`.
- Лексема (token) — наименьшая и неделимая единица языка. Примеры: все идентификаторы, ключевые слова, а также литералы типа 5.2 и `"Привет всем!"`.
- Знак операции (operator) — лексемы, представляющие собой встроенные операторы языка. Примеры: `=`, `+`, `-`, `*`, `/` (оператор присваивания и арифметические операторы). Операции нередко называют операторами.
- Выражение (expression) — группа лексем (обычно литералов и идентификаторов) в сочетании со знаками операций, для которых можно вычислить значение. Примеры: `3.141`, `"Привет!"`, `(2 + x)*5`, `myfunc()`, `x = y + myfunc()`.

Простак. Все эти определения нелегко запомнить.

Профессор. Ничего, со временем, при чтении литературы по программированию и обсуждении программ с коллегами и друзьями вы научитесь правильно употреблять перечисленные термины. Однако можно вполне обойтись и только такими понятиями, как идентификатор (имя), ключевое слово, значение и оператор (операция).

2.7. Резюме и напутствие

Профессор. Итак, мы рассмотрели наиболее важные понятия, присущие большинству современных языков программирования: переменные, типы данных, массивы, функции, классы и объекты, операторы и выражения. Давайте вспомним их краткие определения.

- Переменные — это контейнеры для хранения данных, называемых значениями переменных; переменная имеет имя или идентификатор, чтобы отличить ее от других переменных и обращаться к ней по имени для получения содержащегося в ней значения.
- Значения — могут быть различных типов: числового, символьного (строкового), логического и др. Тип — множество допустимых значений — может быть задан явным перечислением элементов, либо посредством формулировки некоторого ограничения или правила. Так например, логический тип данных есть множество из двух элементов (true и false), интерпретируемых как "истина" и "ложь" соответственно. Множество чисел бесконечно и не может быть задано перечислением своих элементов. Поэтому числовой тип определяют неким

правилом формирования относящихся к нему элементов: последовательность цифр, перед которой может быть указан знак, а разделителем целой и дробной частей служит точка.

- Переменная — принимает значение или, другими словами, в переменную записывается значение с помощью оператора присваивания. Доступ к значению переменной осуществляют по ее имени (идентификатору).
- Массив — упорядоченный набор (последовательность данных). Можно сказать, что конкретный массив это значение составного типа, который также называется массивом.
- Функция — это поименованный блок программного кода (тело функции), который можно вызвать для выполнения из основной (внешней по отношению к коду функции) программы.
- Класс — описание объекта в виде набора его свойств, которые аналогичны обычным переменным и функциям. Часто свойства-переменные называют просто свойствами, а свойства-функции — методами. С формальной точки зрения класс — это контейнер для хранения данных о свойствах объекта. С помощью специальной функции (конструктора), в программе создается экземпляр класса. Свойства и методы экземпляра класса могут быть доступными из внешней программы.
- Операторы — предназначены для задания команд, которые должен выполнить компьютер. Выражения содержат один или несколько операторов. Наиболее часто встречается оператор присваивания значения переменной. Для управления ходом вычислений служат условные операторы и циклы. Если бы не эти специальные операторы, программа выполнялась только в одном направлении — сверху вниз, в порядке упоминания выражений в листинге (исходном тексте).

Далее мы рассмотрим эти и другие понятия программирования на примере конкретных языков — JavaScript и PHP. Вы увидите, что между ними очень много общего, что позволит вам освоить их одновременно. Тем не менее, между ними есть и существенные различия, которыми нельзя пренебречь. Если вам удастся без особых проблем изучить оба этих языка, то, возможно, вы обретете способность к изучению и других (более сложных) языков программирования, таких как C, Java или Pascal. Базовые понятия, с которыми вы уже познакомились в общих чертах при изучении конкретных языков или при чтении специальной литературы, могут воплощаться по-разному. Одно дело теория, другое — конкретная реализация в том или ином программном продукте, на той или иной платформе. Я лишь предупреждаю вас, чтобы вы не паниковали, обнаружив что-то такое, что не работает у вас. В действительности все программисты преодолевают, так или иначе, возникшие трудности, связанные с особенностями операционных систем, исполнительных систем и приложений. Если вы, пользователи компьютера или просто читатели, столкнулись с чем-то, что не работает на вашем компьютере, хотя вы и сделали все, что указывалось в рекомендациях, то, скорее всего, дело в мелочах, которые разрешаются легко и просто. Но иногда возможны и более существенные осложнения, устранение которых лучше поручить специалисту-диагносту. Врачи и программисты знают, как по поверхностным (клиническим) симптомам можно

судить о далеко идущих внутрь организма причинах неполадок, а также о том, какое, хотя бы в общих чертах, назначить лечение (устранение неполадок или ошибок). Да, все это не просто, но вполне преодолимо со временем, было бы желание. Но для начала важно соблюсти некоторые наиболее общие условия.

Освоение языка совершенно не эффективно без практического программирования, как невозможно научиться говорить на иностранном языке вне естественной среды общения. Для вящей убедительности моего тезиса вспомните, что ребенок сначала учится говорить на примерах от своих родителей, а только потом знакомится с правилами языка, которые ему надлежит применять везде и всегда, если только он не хочет поселиться в джунглях и там продолжить свой род. Поэтому необходимо основательно подготовиться к практическому выполнению предлагаемых далее примеров.

Применительно к JavaScript и PHP это совсем не сложно. Программные коды можно писать в обычном текстовом редакторе, например, блокноте Windows.

Чтобы просматривать и отлаживать *клиентские сценарии* (т. е. сценарии, исполняемые Web-браузером) на JavaScript, в вашей системе ничего не нужно устанавливать предварительно, коль скоро у вас уже имеется современный браузер, например, Microsoft Internet Explorer, Mozilla или Opera. Однако следует учитывать, что результат выполнения сценариев может ощутимо зависеть от конкретного типа браузера и его версии.

Для создания и отладки серверных сценариев на языке PHP потребуется программное обеспечение Web-сервера, например, Microsoft Internet Information Services (IIS — Информационные службы Интернета) или Apache, а также установка на сервере модуля PHP. Разумеется, можно воспользоваться удаленным Web-сервером с уже установленным модулем PHP, но изучение языка и отладка сценариев удобнее на локальном компьютере. Установить и настроить Web-сервер с модулем PHP на локальном компьютере может обычный пользователь, а не только системный администратор. Как это сделать, рассказывается в *приложении 2*.