Классика программирования

Кауфман В. Ш.

Языки программирования. Концепции и принципы

ДМК ПРЕСС
ИЗДАТЕЛЬСТВО

УДК 519.682.1
ББК 004.438
К30

Рецензент О. Н. Перминов

К30 Кауфман В. Ш.
Языки программирования. Концепции и принципы. – М.: ДМК Пресс, 2010. – 464 с.: ил.

ISBN 978-5-94074-622-5

Рассмотрены фундаментальные концепции и принципы, воплощенные в современных и перспективных языках программирования. Представлены разные стили программирования (операционный, ситуационный, функциональный, реляционный, параллельный, объектно-ориентированный). Базовые концепции и принципы рассмотрены с пяти различных позиций (технологической, авторской, математической, семиотической и реализаторской) и проиллюстрированы примерами из таких языков, как Паскаль, Симула-67, Смолток, Рефал, Ада, Модула-2, Оберон, Оккам-2, Турбо Паскаль, C++ и др.

Сложность выделена как основополагающая проблема программирования, а абстракция-конкретизация и прогнозирование-контроль – как основные ортогональные методы борьбы со сложностью. На этой общей базе в книге впервые представлена цельная система концепций и принципов, создающая четкие ориентиры в области языков программирования. На основе этой системы сформулированы оригинальные положения, указывающие перспективы развития в этой области (модули исключительных ситуаций, модули управления представлением, входовые типы и др.). Многие из них в последние годы стали реальностью.

Новые подходы применены при изложении известных фактов (пошаговая модификация нормальных алгоритмов Маркова сначала до Рефала, а затем до реляционных языков, сопоставление принципов «сундука» и «чемоданчика» при создании Ады, Модулы-2 и Оберона, развитие концепции наследуемости от модульности до объектной ориентации, систематическое сопоставление концепции параллелизма в Аде и Оккаме-2, и др.).

Для всех, серьезно интересующихся программированием, в том числе научных работников, программистов, преподавателей и студентов.

Ил. 5 Библиогр. 64 назв.

УДК 519.682.1
ББК 004.438

ISBN 978-5-94074-622-5

Содержание

Глава 1

Концептуальная схема языка программирования

1.1. Что такое язык программирования

Для начала дадим экстенсиональное определение ЯП – явно перечислим те конкретные языки, которые нас заведомо интересуют (их мы уверенно считаем языками программирования). Это Фортран, Симула, Паскаль, Бейсик, Лисп, Форт, Рефал, Ада, Си, Оккам, Оберон. Однако хочется иметь возможность на основе определения предсказывать новые частные случаи, в определении не перечисленные. Такое определение должно опираться на существенные свойства выбираемых для изучения языков – оно должно быть интенсиональным. Дадим одно из возможных интенсиональных определений ЯП.

Язык программирования – это инструмент для планирования поведения исполнителя.

Однако, во-первых, перечисленные выше ЯП служат не только для планирования поведения исполнителей (компьютеров), но и для обмена программами между людьми. Такая важнейшая функция существенно влияет на устройство и принципы создания ЯП (хотя она все же вторична – можно показать, что люди должны понимать и читать программы, даже не имея никаких намерений ими обмениваться; просто иначе достаточно крупной программы не создать). Эту функцию языка никак нельзя игнорировать при изучении ЯП.

Во-вторых, в нашем определении каждое слово нуждается в уточнении. Являются ли «инструментами для планирования поведения исполнителя» должностная инструкция, письменный стол, переговорное устройство, правила уличного движения, русский язык?

1.2. Метауровень

Взглянем на наши действия с позиции стороннего наблюдателя, отвлечемся от своей роли соответственно автора и читателей на только что законченном начальном отрезке нашей совместной работы. Другими словами, поднимемся на метауровень, чтобы обозревать исходный уровень в целом.

Чем мы занимались?

Во-первых, попытались добиться взаимопонимания в вопросе о том, что такое ЯП. Во-вторых, начали применять для достижения взаимопонимания метод последовательных уточнений.

Чего мы добились и что осталось неясным? Стало яснее, что будем изучать, – можем привести примеры ЯП, с которыми все согласны, и указать объекты, заведомо не являющиеся ЯП в соответствии с нашим определением (также рассчитывая на общее согласие), скажем, левая тумба письменного стола. Почувствовали, что добиться взаимопонимания (даже по поводу привычных понятий) очень непросто. Осталось неясным, в частности, с какой позиции и с какой целью мы намерены изучать ЯП.

Постараемся в первом приближении устранить эти неясности. Однако заниматься последовательными уточнениями многих важных понятий мы будем на

протяжении всей нашей работы – ведь она не формальная, а содержательная, нас интересуют реально существующие, возникающие на наших глазах и развивающиеся объекты – живые ЯП. Поэтому-то и невозможно дать исчерпывающего описания ЯП как понятия (это понятие живет вместе с нами).

Начнем с более внимательного рассмотрения преград, обычно возникающих на пути к взаимопониманию.

1.3. Модель передачи сообщения

Добиться взаимопонимания бывает очень сложно. Чтобы выделить возникающие здесь проблемы, рассмотрим следующую модель передачи сообщения (рис. 1.1).

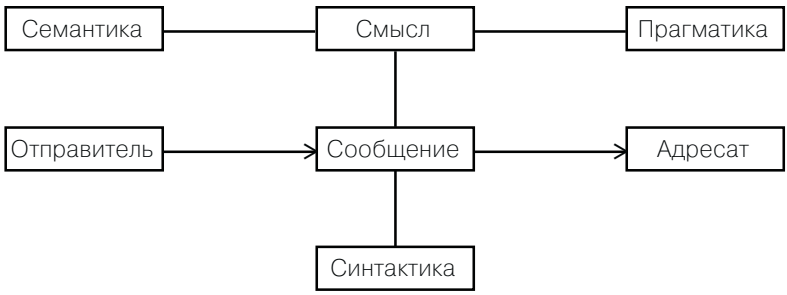

Рис. 1.1

В этой модели выделены понятия «отправитель» (автор, генератор сообщения), «адресат» (получатель, читатель, слушатель сообщения), собственно «сообщение» (текст, последовательность звуков), «смысл» сообщения (нечто обозначаемое сообщением в соответствии с правилами, известными и отправителю, и адресату).

Выделены также названия наук, занимающихся соответственно правилами построения допустимых сообщений (синтактика), правилами сопоставления таким сообщениям смысла (семантика) и правилами, регулирующими использование сообщений (прагматика).

1.4. Классификация недоразумений

С помощью модели на рис. 1.1 займемся классификацией недоразумений, возникающих при попытке установить взаимопонимание.

Автор (отправитель сообщения) может подразумевать одну структуру сообщения, а адресат (получатель) – другую, как в классическом королевском указе: «Казнить нельзя помиловать!» Это синтаксическое недоразумение.

Автор может употребить слово с неточным смыслом, подразумевая один его оттенок, а адресат выберет другой. Рассмотрим, например, фрагмент рецепта при-

готовления шоколадной помадки: «изредка помешивая, варить на слабом огне до тех пор, пока капля не станет превращаться в холодной воде в мягкий шарик». Не потому ли кулинарное искусство и является искусством, а не точной наукой, что разные повара, зная один и тот же рецепт, по-разному понимают слова «изредка», «медленно», «холодной», «мягкий», а также с разной частотой станут пробовать «превратить каплю в мягкий шарик». Естественно, и результаты у них будут разные. Это семантическое недоразумение.

Наконец, автору трудно иногда представить себе, какую интерпретацию может придать его сообщению адресат, если у них сильно различаются представления о мире или решаемые задачи.

Например, сообщение лектора о предстоящем коллоквиуме может быть воспринято студентами как призыв не посещать малоинформативные лекции, чтобы иметь время для работы с книгами. Это уже прагматическое недоразумение.

Нетрудно привести и другие примеры синтаксических, семантических и прагматических недоразумений при попытке достичь взаимопонимания.

Почему же в самом начале речь пошла о взаимопонимании и о стоящих на пути к нему преградах? В основном по двум причинам.

Во-первых, ЯП – это инструмент для достижения взаимопонимания (безопаснее «взаимопонимания») людей с компьютерами и между людьми по поводу управления компьютерами. Поэтому в принципах построения, структуре, понятиях и конструктах ЯП находят свое отражение и сущность общей проблемы взаимопонимания, и взгляды творцов ЯП на эту проблему, и конкретные методы ее решения.

Во-вторых, способ, которым люди преодолевают преграды на пути к взаимопониманию, содержит некоторые существенные элементы, остающиеся важными и при общении с компьютерами (в частности, при создании и использовании ЯП). Кстати, в Международной организации по стандартизации ИСО разработан документ [1], регламентирующий устройство стандартов ЯП. Он содержит классификацию программных дефектов, полностью согласующуюся с нашей классификацией недоразумений.

Особенно бояться синтаксических недоразумений не стоит. Они касаются отдельных неудачных фраз и легко устраняются немедленным вопросом (устным или письменным). В ЯП это тоже не проблема – таких недоразумений там просто не бывает. Дело в том, что создатели ЯП руководствуются принципом однозначности: **язык программирования должен быть синтаксически однозначным** (то есть всякий правильный текст на ЯП должен иметь единственную допустимую структуру).

Итак, сформулирован один из принципов построения ЯП, отличающих их, например, от языков естественных. Такого рода общие принципы и концепции нас и будут интересовать в первую очередь.

Семантические недоразумения опаснее. Если, скажем, слово «язык» будет ассоциироваться с субпродуктом, ставшим весьма редким гостем прилавка, то недоразумение может не ограничиться пределами одной фразы. Большой язык, свежий язык, красный язык, зеленый и голубой язык – все это может касаться и го-

вяжьего языка, и ЯП (в конкурсе языковых проектов, ставшем одним из этапов создания языка Ада, языки-конкуренты получили условные «цветные» наименования; победил «зеленый» язык).

Метод борьбы с семантическими недоразумениями при человеческом общении известен – нужно выделять важные понятия, давать им четкие определения, приводить характерные примеры. Это со стороны говорящего. Слушатели должны, в свою очередь, стараться уловить оставшиеся существенные неясности, приводить контрпримеры (объектов, соответствующих определениям, но, по-видимому, не имевшихся в виду говорящим, и объектов, не соответствующих определениям, но, скорее всего, имевшихся в виду). Этот же метод точных определений широко используется в ЯП (определения процедур, функций и типов в Паскале), а примеры и контрпримеры применяются, как известно, при отладке программ.

1.5. Отступление об абстракции-конкретизации. Понятие модели

Добиваясь взаимопонимания, мы активно пользуемся аппаратом абстракции-конкретизации (обобщения-специализации).

Создавая понятие, отвлекаемся (абстрагируемся) от несущественных свойств тех конкретных объектов, на основе знания которых понятие создается, и фиксируем в создаваемом понятии лишь свойства существенные, важные с точки зрения задачи, решаемой с применением этого понятия. Так, в понятии «часы» мы обычно фиксируем лишь свойство быть «устройством, показывающим время», и отвлекаемся от формы, структуры, цвета, материала, изготовителя и других атрибутов конкретных часов.

Приводя пример, мы конкретизируем абстрактное понятие, «снабжая» его второстепенными с точки зрения его сущности, но важными в конкретной ситуации деталями. Так, конкретное выполнение процедуры происходит при конкретных значениях ее параметров; у конкретного примера ЯП – Фортрана – конкретный синтаксис и конкретная семантика.

Мои часы большие, круглые, позолоченные, с тремя стрелками, марки «Восток», на 17 камнях. Все это немного говорит об их качестве в роли «устройства, показывающего время», но конкретное устройство всегда обладает подобными «необязательными», с точки зрения его роли, свойствами. Их существование лишь подчеркивает тот факт, что (абстрактное) понятие никогда не исчерпывает конкретного объекта – оно всегда отражает лишь некоторую точку зрения на этот объект, служит компонентой его модели, оказавшейся удобной для решения определенной задачи. В другой ситуации, при решении другой задачи, этот же конкретный объект может играть другую роль. Тогда и точка зрения на него может быть другой, и может потребоваться совсем другая модель того же самого объекта.

На «устройство, показывающее время», в известных условиях можно предпочесть смотреть как на «украшение», и с этой точки зрения (в этой его роли) важнее станут форма, цвет, размер, фирма, важнее даже способности правильно пока-

зывать время. На процедуру можно смотреть как на «объект, расходующий машинные ресурсы». При такой ее роли совершенно не важно, каков смысл выполняемых в ней действий. На ЯП иногда приходится смотреть как на объект стандартизации, и тогда важно не столько то, каковы именно особенности его семантики и синтаксиса, сколько то, найдется ли достаточно много заинтересованных в тех или иных его свойствах людей и организаций.

Непроизвольная, а иногда и намеренная, но не подчеркнутая явно смена точки зрения, переход по существу к другой модели объекта мешает взаимопониманию, служит источником прагматических недоразумений. Вы говорите, что часы «плохие», потому что некрасивые, а я говорю «хорошие», так как они отлично работают.

Способность без затруднений переходить от одной модели к другой, четко фиксировать и легко изменять уровень рассмотрения, а также угол зрения, отмечается обычно как важнейшее профессиональное качество программиста.

1.6. Синтактика, семантика, прагматика

Устранять прагматические недоразумения бывает особенно сложно, когда они связаны с различием не только точек зрения, но и целевых установок. Если правильно разложить фразу на составляющие может помочь согласование с контекстом, а правильно понять смысл слова или фразы может помочь знание их назначения (роли), то восстановить эту роль, догадаться о ней, если об этом не сказано явно, очень тяжело. Слишком велика неопределенность, свобода выбора.

Представим себе положение человека, которому излагается последовательность определений и не говорится, зачем они вводятся, для решения каких задач предназначены. Это хорошо знакомая всем ситуация – есть такой стиль изложения математических результатов. Слушатель (читатель) при этом лишен всякой опоры для контроля, кроме поиска чисто логических противоречий. Пока он не понял, зачем все это нужно, он может пропустить любую содержательную ошибку. А попробуйте понять смысл программы, если неизвестно, для чего она написана!

Вывод очевиден – для достижения взаимопонимания необходимо, чтобы отправитель и адресат пользовались, во-первых, одинаковыми правилами разложения сообщения на составляющие (изучением таких правил занимается синтактика); во-вторых, согласованными правилами сопоставления сообщению смысла (такими правилами занимается семантика); в-третьих, имели согласованные целевые установки (это предмет прагматики).

Ролью перечисленных аспектов для создания и использования ЯП мы еще займемся, а сейчас уместно поговорить об основной цели книги (для согласования наших целевых установок).

Мы намерены изложить принципы оценки, создания и использования современных ЯП. Это очень нужная, плодотворная и увлекательная, но далеко не устоявшаяся, быстро развивающаяся область. Поэтому нет возможности опираться на освященный традицией опыт предшественников, а также стабильные программы

и учебники (как это бывает, скажем, при изучении математического анализа или дифференциальных уравнений). Приходится рисковать и экспериментировать.

Итак, о нашей основной цели. Она состоит в том, чтобы постараться правильно ориентировать читателя в области ЯП, помочь ему осознать навыки и опыт, приобретенные при самостоятельной работе с конкретными ЯП.

Но не слишком ли опасна идея «правильно» ориентировать? Ведь если, скажем, представления автора о профессиональных запросах читателя или о тенденциях развития ЯП окажутся ошибочными, то скорее всего «правильная» ориентация на самом деле окажется дезориентацией. Не лучше ли ограничиться изложением бесспорных положений из области ЯП – уж они-то понадобятся наверняка?!

К сожалению или к счастью, альтернативы у нас, по сути, нет. Абсолютно бесспорные положения касаются, как правило, лишь конкретных ЯП. Например, «Один из операторов в языке Паскаль – оператор присваивания. Устроен он так-то. Служит для того-то». В хорошо известном учебнике программирования это положение обобщено. Сказано так: «Фундаментальным действием в любом алгоритмическом языке является присваивание, которое изменяет значение некоторой переменной». И это уже неверно! Сейчас много внимания уделяется так называемому функциональному программированию, аппликативным ЯП, где присваивание – не только не «фундаментальное» действие, но его вообще нет!

Значит, в области ЯП нет достаточно общих бесспорных положений? В некотором смысле есть. Чаще не столь бесспорных, сколь заслуживающих изучения. Правда, их общность несколько другого характера. Примером может служить упоминавшийся принцип однозначности. Да и приведенная фраза из учебника – вполне бесспорное положение, если считать, что она характеризует определенный класс ЯП, в который не попадает, скажем, язык Лисп – один из самых «заслуженных», распространенных и в то же время перспективных. Итак, даже если ограничиться лишь относительно бесспорными положениями, их все равно нужно отбирать с определенных позиций, с определенной целью. Естественная цель – стремиться принести читателю максимальную пользу. Опять мы приходим к «угадыванию» будущих потребностей.

1.7. Зачем могут понадобиться знания о ЯП

Во-первых, каждая программа должна общаться (обмениваться информацией) с внешним миром. Соглашения, определяющие способ общения, – это язык, так что понимание принципов построения языков – необходимая компонента грамотного программирования. Исключительно важная компонента, потому что непосредственно связана с внешним эффектом программы, со способом ее использования. При разработке внешнего сопряжения своей программы программист обязан проявить истинный профессионализм, предоставляя пользователю максимум услуг при минимуме затрат. Особенно это важно при создании пакетов приклад-

ных программ, инструментальных систем, вообще любых программных изделий, предназначенных для эксплуатации без участия автора.

Во-вторых, каждый ЯП – это своя философия, свой взгляд на деятельность программиста, отражение определенной технологии программирования. Даже представлений об Алголе-60, Фортране и Бейсике достаточно, чтобы почувствовать, что имеется в виду.

Скажем, творцы Алгола (выдающиеся представители международного сообщества ученых в области информатики под руководством Петера Наура) с естественным для них академизмом придавали относительно много значения строгости определения и изяществу языковых конструктов. Считалось, что самое важное в работе программиста – сформулировать алгоритм (и, возможно, опубликовать его). Переписать программу в расчете на конкретные устройства ввода-вывода считалось не заслуживающей особого внимания технической деятельностью. Не привлек должного внимания авторов языка и такой «технический» аспект программистской деятельности, как компоновка программ из модулей.

Творцы Фортрана (сотрудники фирмы IBM во главе с Джоном Бэкусом) в значительной степени пренебрегли строгостью и изяществом языка и со свойственным им в ту пору (1954–1957 гг.) прагматизмом уже в первых версиях языка уделили особое внимание вводу-выводу и модульности. Но ни Фортран, ни Алгол не рассчитаны на работу в диалоговом режиме, в отличие от Бейсика (созданного в Дартмундском колледже первоначально для обучения студентов).

Таким образом, изучение ЯП дает знание и понимание разнообразных подходов к программированию. Это полезно при любой программистской деятельности.

В-третьих, понимание общих принципов и концепций, определяющих строение и применение ЯП, позволяет легче и глубже освоить конкретный язык – основной профессиональный инструмент программиста.

В-четвертых, и это хотелось бы подчеркнуть особо, понятия и тенденции в области ЯП с некоторым запаздыванием (в целом полезным) довольно точно отражают понятия и тенденции собственно программирования как науки, искусства и ремесла (и просто области человеческой деятельности). В этом смысле мы обсуждаем основные принципы и понятия программирования, но со специфически языковой точки зрения.

Все, о чем было сказано до сих пор, касалось интересов потенциального пользователя ЯП. Но читатель может оказаться и руководителем коллектива, которому требуется оценивать и выбирать ЯП для выполнения конкретного проекта (учитывать, скажем, затраты на освоение этого ЯП или на обмен написанными на нем программными изделиями). Если же он станет творцом ЯП, создателем транслятора или руководства для пользователей, то ему понадобятся столь разнообразные знания о ЯП, что их придется извлекать целеустремленным изучением специальной литературы. Можно надеяться дать лишь первоначальный импульс в нужном направлении.

Конечно, предсказать, для чего именно понадобятся приобретенные знания, сложно. Могут напрямую и вовсе не понадобиться. Но наверняка пригодится приобретенная обсуждениями, размышлениями и упражнениями культура рабо-

ты со сложными объектами при решении сложных задач. В нашем случае это такие задачи, как оценка, использование, разработка и реализация ЯП.

1.8. Принцип моделирования ЯП

Было бы неправильно ставить нашей целью научить свободному владению конкретными ЯП, пусть даже особо привлекательными или перспективными. Для этого служат специальные учебники, упражнения и, главное, практика.

Наша задача – познакомить с важнейшими понятиями и концепциями, помогающими оценивать, использовать, реализовывать и разрабатывать ЯП, дать представление о направлениях и проблемах их развития. Поэтому займемся в основном изучением моделей ЯП. Другими словами, при изучении ЯП будем систематически применять принцип моделирования (как самих реальных ЯП, так и их отдельных аспектов). Наиболее важные по тем или иным причинам ЯП или их конструкты иногда будут рассмотрены довольно подробно, но прежде всего лишь как примеры, иллюстрирующие более общие положения.

Например, важно понимать, что с каждым ЯП связан эталонный (абстрактный) исполнитель, в котором, в свою очередь, определены данные, операции, связывание, именование, аппарат прогнозирования и контроля, а возможно, и аппарат исключений, синхронизации и защиты. Важно понимать перечисленные термины, назначение соответствующих языковых конструктов и уметь ими пользоваться при решении практических задач. Но не очень важно помнить наизусть все связанные с ними тонкости в конкретных ЯП. Последнее может оказаться важным лишь тогда, когда тонкости иллюстрируют ключевые концепции рассматриваемого ЯП. Например, жесткие правила выбора обозначений в Бейсике непосредственно связаны с его ориентацией на относительно небольшие программы и простоту реализации.

1.9. Пять основных позиций рассмотрения ЯП

Итак, будем считать, что целевые установки согласованы в достаточной степени, чтобы сделать следующий шаг – приступить к систематическому изучению нашего предмета.

И сразу вопрос – с чего начать? Легко сказать «систематическому». Но ведь системы бывают разные. Часто начинают «снизу» – с основных конструктов, встречающихся почти во всех существующих ЯП. Тогда мы сразу погружаемся в мир переменных, констант, параметров, процедур, циклов и т. п. Такой путь привлекателен хотя бы тем, что им сравнительно легко пойти. Но на этом пути за деревьями обычно не видно леса, не удается увидеть ЯП в целом, построить его адекватную модель.

Поэтому выберем другой путь. Постараемся взглянуть на объект нашего изучения – ЯП – с общих позиций. Нас будут особенно интересовать технологическая, семиотическая и авторская позиции.

Первая названа технологической потому, что отражает взгляд человека, желающего или вынужденного пользоваться ЯП как технологическим инструментом на каком-либо из этапов создания и использования программных изделий (другими словами, в течение их жизненного цикла). С таким человеком естественно объясняться в технологических терминах.

Вторая позиция названа семиотической потому, что ее можно представить себе как позицию человека, знакомого с некоторыми знаковыми системами (русским языком, дорожными знаками, позиционными системами счисления) и желающего узнать, чем выделяются такие знаковые системы, как ЯП. Следует объяснить ему это в семиотических терминах.

Третья позиция – авторская. Автор создает ЯП, делает его известным программистской общественности, исправляет и модифицирует его с учетом поступающих предложений и критических замечаний.

Уделим внимание и другим позициям – математической и реализаторской.

Математик понимает, что такое математическая модель изучаемого объекта, и желает познакомиться с математическими моделями ЯП. С ним желательно объясняться в математических терминах.

Реализатор обеспечивает возможность пользоваться ЯП как средством практического программирования. Другими словами, он не только создает трансляторы, но и пишет методические руководства, обучающие и контролирующие программы, испытывает трансляторы и т. п.

Уместно подчеркнуть, что с разных позиций мы будем рассматривать один и тот же объект. Начнем с технологической позиции. Установим связь ЯП с производством программных услуг.

1.10. Что такое производство программных услуг

Напомним исходные понятия, известные из общего курса программирования компьютеров. Понятие компьютер нужно уточнить лишь в той мере, в которой это необходимо для нашей цели. Важно, что компьютер обладает двумя фундаментальными способностями – хранить данные и выполнять планы.

Начнем со второй способности. Назовем исполнителем всякое устройство, способное выполнять план. Так что и компьютер, и робот, и рабочий, и солдат, и сеть компьютеров, и коллектив института способны играть роль исполнителя.

Всего одна фундаментальная способность – выполнять план – дает возможность исполнителю предоставить пользователю содержательно разнообразные услуги. Определяя конкретные планы, можно настраивать исполнителя на предоставление конкретных услуг. Важно лишь, чтобы в плане фигурировали задания, посильные для выбранного исполнителя. Посильные – это значит такие, для выполнения которых у исполнителя имеются соответствующие ресурсы.

Для компьютеров как исполнителей характерны два вида ресурсов – память и процессор. Память реализует первую из двух названных фундаментальных способностей – служит для хранения данных. Это пассивный ресурс. Процессор реа-

лизует вторую из названных способностей – служит для выполнения действий, предусмотренных в планах. Это активный ресурс. Процессор характеризуется определенным набором допустимых действий (операций, команд). Действия из этого набора считаются элементарными (в плане не нужно заботиться о способе выполнения таких действий).

Две названные способности связаны – выполнение достаточно сложного плана требует его хранения в доступной исполнителю памяти. В свою очередь, реализация хранения требует способности выполнять план (данные нужно размещать, перемещать, делать доступными).

План для такого исполнителя, как компьютер, должен в итоге сводиться к указанию конечной последовательности элементарных действий. Такой план называют программой.

> Людей как исполнителей характеризует прежде всего наличие у них модели реального мира, в достаточной степени согласованной с моделью мира у создателя плана. Поэтому в плане для людей можно указывать цели, а не элементарные действия.

Ресурс, существенный почти для всех реальных исполнителей, – это время. Важное свойство компьютеров как исполнителей – способность выполнять элементарные действия исключительно быстро (порядка микросекунды на действие). Не менее важное свойство компьютера – способность хранить огромные объемы данных (в оперативной памяти – мегабайты; на внешней – практически не ограничено).

Именно способность компьютеров выполнять весьма длинные последовательности элементарных действий над данными любого нужного объема за практически приемлемое время предоставляет пользователям весьма разнообразные по содержанию услуги (развлекать, играя с ним в интеллектуальные и (или) азартные игры, давать справки, помогать в составлении планов, управлять самолетами и танками, поддерживать светскую беседу).

Чтобы настроить компьютер на конкретный вид услуг, нужно снабдить его соответствующими этому виду услуг знаниями в приемлемой для него форме. Принципиально важный для нас факт, ставший очевидным лишь относительно недавно (по мере расширения и развития сферы предоставляемых компьютерами услуг), состоит в том, что самое сложное (и дорогое) в этом деле – не сами компьютеры, а именно представление знаний в приемлемой для них форме. В наше время аппаратура в компьютере по своей относительной стоимости иногда сравнима с упаковкой, в которую «заворачивают» знания.

Знания, представленные в компьютерах, традиционно называют программами (таким образом, под программой в широком смысле слова понимают не только планы). Соотношение стоимости аппаратуры и программ (а также иные соображения) во многих случаях дает основание абстрагироваться от используемой аппаратуры и говорить об услугах, предоставляемых непосредственно программами, а также о производстве программных услуг.

Подчеркнем, что пока далеко до полной независимости программ от используемой аппаратуры. Проблема переноса программ (на исполнитель другого типа) – одна из острейших проблем современного программирования. И тем не ме-

нее обычно затраты на создание достаточно сложной программы определяются в основном сущностью решаемой задачи (предоставляемой услуги), а не используемой аппаратуры.

Правомерность абстракции от аппаратуры подчеркивает определенную искусственность проблемы переноса программ. Если пока в значительной степени безразлично, на чем программировать, то разнообразие и несовместимость исполнителей вызваны не объективными, а социальными причинами.

Знания, представляемые в компьютерах, можно разделить на пассивные и активные. Первые отражают факты, связи и соотношения, касающиеся определенного вида услуг. Вторые – это рецепты, планы действий, процедуры, непосредственно управляющие исполнителем.

Представление пассивного знания ориентировано в первую очередь на такой ресурс компьютера, как память, а представление активного – на процессор. Исторически самыми первыми сферами применения компьютеров оказались такие, где главенствовало активное знание – эксплуатировалась способность ЭВМ не столько много знать, сколько быстро считать.

Кстати, и память на первых ЭВМ была очень мала по сравнению со скоростью работы процессора. Всю свою память они могли просмотреть (или переписать) за десятую долю секунды. Недалеко ушли в этом отношении и современные компьютеры (если говорить об оперативной памяти).

Соотношение между объемом оперативной памяти и скоростью процессоров активно влияет на мышление программистов, инженеров, пользователей, а также всех связанных с компьютерами людей. Изменение этого соотношения (а его разумно ожидать) способно произвести революцию в практическом программировании (см. далее о модифицированной модели Маркова и функциональном стиле программирования (модель Бэкуса); есть и другие перспективные модели, например реляционная). Пока же программисты вовсю экономят память, помещая новые значения на место старых, а преподаватели учат искусству такого «эффективного» программирования. В «эффективных» программах трудно восстановить процесс получения результата, трудно объяснить неожиданный результат. Традиционный стиль программирования, обусловленный бедностью ресурсов, затрудняет написание, понимание, проверку и удостоверение правильности программ. Тенденция развития состоит в том, что роли активного и пассивного знаний в производстве программных услуг становятся более симметричными, а забота об эффективности отступает перед заботой о дружественности программы.

1.11. Производство программных услуг – основная цель программирования

Измерять эффективность того или иного производства разумно лишь по отношению к его цели (конечному результату). Поэтому важно понимать, что конечная цель программирования – не создание программ самих по себе, а предоставление

программных услуг. Другими словами, программирование в конечном итоге нацелено на обслуживание пользователей. А настоящее обслуживание должно руководствоваться известным принципом: «Клиент всегда прав». В применении к программированию этот принцип означает, что программы должны быть «дружественными» по отношению к пользователю. В частности, они должны быть надежными, робастными и «заботливыми».

Первое означает, что в программе должно быть мало ошибок, второе – что она должна сохранять работоспособность в неблагоприятных условиях эксплуатации, третье – что она должна уметь объяснять свои действия, а также ошибки пользователя.

Что значит «мало ошибок», зависит от назначения программы (ясно, что программа обучения русскому языку и программа автопилота могут иметь различную надежность). Под «неблагоприятными условиями» понимается ограниченность выделяемых программе ресурсов (память, каналы ввода-вывода, число процессоров), перегрузка (много пользователей, большие объемы данных), ошибки пользователей, сбои и отказы аппаратуры, попытки намеренно вывести программу из строя и т. п.

Сказанное относится к работе программы. Однако важно понимать, что программная услуга – это не только услуга, оказываемая пользователю непосредственно при работе компьютера под управлением программы, но и проверка, оценка, продажа, подбор программ, их перенос на другой исполнитель, сопровождение и аттестация (авторитетное подтверждение качества) программ и т. п.

Когда производство программных услуг достигает некоторой степени зрелости, из кустарного производства вырастает индустрия программных услуг и обслуживающая ее потребности теория программирования. Как индустрия, так и кустарное производство пользуются при этом той или иной технологией – технологией производства программных услуг, то есть технологией программирования.

(О связи науки, искусства, теории и технологии в программировании см. замечательную Тьюринговскую лекцию **Дональда Кнута** в Communication of the ACM. – 1974. – Vol. 12. – P. 667–673).

1.12. Сложность как основная проблема программирования

Итак, основная цель программирования – производство программных услуг. Известно, что этот род человеческой деятельности в развитых странах уверенно выходит на первое место среди всех других производств (скажем, в США соответствующая отрасль хозяйства уже опередила недавних лидеров – нефтяную и автомобильную отрасли).

Вместе с тем известно, что создание программ и предоставление других связанных с ними услуг остается слишком дорогим и относительно длительным делом, в котором трудно гарантировать высококачественный конечный результат.

В чем же основная причина такого положения? Связана ли она с самой природой программирования или носит субъективный характер?

В настоящее время краткий ответ можно сформулировать так: «сложность – основная проблема программирования; связана с самой его природой; можно надеяться на ее понижение для освоенных классов задач».

1.13. Источники сложности

Попытаемся подробнее разобраться с тем, почему же сложность объектов, с которыми приходится иметь дело, – отличительная черта программирования компьютеров. Найдя источники сложности, можно с большей надеждой на успех искать пути ее преодоления.

Когда исполнители работают медленно, а действия, которые считаются элементарными, специфичны для вида предоставляемых услуг, то планирование деятельности исполнителя и критерии качества такого планирования существенно зависят и от услуги, и от конкретного набора элементарных действий.

Вряд ли стоит поручать, скажем, заводскому технологу, специалисту по обработке металлов резанием, планирование индивидуального пошива верхней одежды. Для этого есть закройщики, которые, в свою очередь, вряд ли смогут применить свое искусство при создании заводских технологических карт. Другими словами, набор «элементарных» действий двух рассмотренных категорий исполнителей считать эквивалентными неестественно.

Компьютеры работают быстро, и наборы их команд в известном смысле эквивалентны. Причем уже в течение многих лет сохраняется тенденция к увеличению скорости и объема памяти при фиксированной цене (примерно на порядок за десятилетие). В таких условиях возникает принципиальная возможность настроить исполнителя на предоставление услуг из очень широкого класса (во всяком случае, границы этого класса неизвестны). Для этого достаточно снабдить исполнителя подходящим планом (написать программу).

Эта принципиальная возможность соблазняет настраивать компьютеры на виды услуг, очень далеких от элементарных возможностей исполнителя. Более того, таковы почти все практически значимые услуги. Поэтому в общем случае план должен предусматривать огромное количество элементарных действий над огромным количеством элементарных объектов. Но этого мало. Самое главное – огромное количество связей между этими объектами. Поэтому и сами программы становятся огромными (уже имеются программы из миллионов команд, например для управления военными и космическими объектами).

Между тем способности человека работать с большим числом связанных объектов, как хорошо известно, весьма ограничены. В качестве ориентира при оценке этих способностей указывают обычно на так называемое «число Ингве», равное семи (плюс-минус 2). Другими словами, человек обычно не в состоянии уверенно работать с объектом, в котором более семи компонент с произвольными взаимными связями. До тех пор, пока программирование остается в основном человеческой деятельностью, с указанным ограничением необходимо считаться.

Таким образом, предоставив универсальность, скорость и потенциально неограниченную память, создатели компьютеров, с одной стороны, соблазнили человечество неслыханными возможностями, а с другой – поставили лицом к лицу с проблемами невиданной потенциальной сложности (при попытке осуществить эти гипотетические возможности).

В этой связи упомянем известный принцип «труд на юзера спихнуть» (*user* – пользователь). Изобретатели компьютеров, предоставив в распоряжение программистов исключительное по своим возможностям абстрактное устройство, «спихнули» на них труд по настройке этого абстрактного устройства на предоставление конкретных услуг. Но такая конкретизация оказалась далеко не тривиальной. Программисты, в свою очередь, создавая «универсальные» программы, «спихивают» труд по их применению в конкретных условиях на потенциальных пользователей этих программ.

Итак, первый источник сложности в программировании – так называемый семантический разрыв – разрыв между уровнем и характером элементарных объектов и операций, с одной стороны, и потенциально возможных услуг – с другой. Иными словами, это проблема согласования масштаба – ювелирными инструментами предлагается сооружать города.

Именно этот источник имелся в виду, когда шла речь об объективном характере присущей программированию сложности. Занимаясь определенным классом услуг (задач), можно стремиться выделить характерный именно для этого класса набор элементарных объектов и операций, построить соответствующий исполнитель (аппаратным или программным способом) и программировать на таком более подходящем исполнителе. Фактически это означает создать адекватный выбранному классу услуг ЯП. На практике это самый распространенный способ борьбы со сложностью и одновременно основная причина роста проблемно-ориентированных языков (ПОЯ).

Имеется еще один принципиально важный источник сложности, затрудняющий «взаимопонимание» с компьютерами. Речь идет об отсутствии в современных компьютерах модели реального мира, согласованной с представлениями о мире у программистов и пользователей. Поэтому в общем случае компьютер не в состоянии контролировать указания программиста или действия пользователя с прагматической точки зрения (контролировать соответствие между действиями и теми целями, ради которых эти действия совершаются, – цели компьютеру неизвестны).

Из-за этого самая «мелкая», с точки зрения создателя программы, описка может привести к совершенно непредсказуемым последствиям (широко известен случай, когда из-за одной запятой в программе на Фортране взорвалась космическая ракета, направлявшаяся на Венеру; пропали усилия, стоившие миллиарды долларов).

Итак, в качестве второго источника сложности в современном программировании следует назвать незнание компьютером реального мира. Лишенный необходимых знаний, компьютер не может не только скорректировать неточно указанные в программе действия, но и проинформировать об отклонениях от направления на