ПРОБЛЕМНЫЕ ЗАДАЧИ ПРИ ИЗУЧЕНИИ ОБЩИХ МЕТОДОВ ИНФОРМАТИКИ И ПРОГРАММИРОВАНИЯ

И.Н. Скопин

Кафедра программирования
Новосибирский государственный университет
ул. Пирогова, 2, Новосибирск, Россия, 630090

Преимущества проблемного подхода к обучению информатике демонстрируются на примере классической задачи, возникающей при разработке компилирующих систем. Два конкурентных способа ее решения позволяют сделать вывод о том, что любой общий метод имеет границы своего применения.

Ключевые слова: стиль программирования, компиляция, синтаксическое управление, информатика, программирование.

Преподавание информатики сегодня зачастую связывается лишь с развитием у обучаемых навыков использования инструментов. Во многих случаях это понимается как изучение офисных инструментов, полезных для подготовки документов, но практически не связанных развитием мышления. Но даже тогда, когда учебный курс предполагает изучение языков и методов программирования, преподавание этих дисциплин чаще всего ограничиваются разработкой простых алгоритмов, их кодированием и выполнением полученных программ. Язык программирования при таком подходе играет роль более или менее удобного инструмента, и если преподаватель стремится к развитию мышления обучаемых, то это связывается лишь с алгоритмикой. В то же время информатика и программирование — особая область творческой деятельности, характерная особенность которой — не достижение принципиальной возможности получения решения, а мотивированный выбор варианта, наилучшего в том или ином смысле. Развитие способности мотивированного выбора полезно далеко за пределами информатики и программирования, оно является важной составляющей логического мышления.

Мы предлагаем такой подход к преподаванию информатики, при котором ставятся проблемные задачи с неоднозначными решениями, предполагающими специальный анализ при выборе варианта для оптимальной реализации. При разборе решений такие задачи позволяют затрагивать вопросы, которые выходят за рамки конкретной ситуации, оказываются основополагающими для дисциплины, органично соединяющими разделенные аспекты в единый предмет. Не менее важно, что изучение основ курса становится мотивированным, а не догматически преподнесенным ученикам как непререкаемая истина.

Тема задачи, демонстрирующей педагогические возможности подхода, — синтаксический анализ как составная часть компилятора. На этом примере показывается, как даже на самых первых этапах изучения информатики и программирования можно дать представление о стилях программирования, которые применимы в разных ситуациях, и, что не менее важно, об условиях, при которых применение стиля перестает быть разумным. Попутно удается продемонстрировать преимущества модульного программирования. Таким образом, рассмотрение хрестоматий-

ной задачи обучения под проблемным углом зрения дает возможность обучаемым понять методы программирования в целом, развивает их образ мышления, полезный как для специализированной подготовки, так и в общеобразовательном плане.

Общие сведения о компиляции. Рассказ о компиляции целесообразно начинать с представления задач компиляции и структуры компилятора. В зависимости от принятого уровня изложения эта часть может строиться по-разному, но в любом случае, готовя учащихся к заключительной проблемной задаче, учитель должен убедиться, что ими освоены понятия компилируемого текста программы и видов анализа при компиляции, объектного кода и его получения из исходной программы.

Удобно исходить из классической структуры компилятора и в связи с этим опираться на понятия, отражающие логическую структуру компилятора:

Front-end — компоненты системы, воспринимающие текст компилируемой программы и преобразующие его во внутреннее для компилятора представление;

Back-end — относительно независимая от языка часть компилятора, предназначенная для построения исполняемого на реальном вычислительном оборудовании объектного кода;

Middle-end (дополнительное понятие, используемое, когда рассказ затрагивает оптимизационные аспекты компиляции) — связующая часть, отвечающая за то, чтобы back-end имел возможность построить более качественный объектный код.

На рисунке 1 представлена классическая схема организации процессов компилятора. Серые блоки указывают на главные задачи Front-end, Middle-end и Back-end. Под ними представлены блоки компонент компилятора, решающих эти задачи. Стрелки с подписями указывают на передачу данных между Front-end, Middle-end и Back-end.

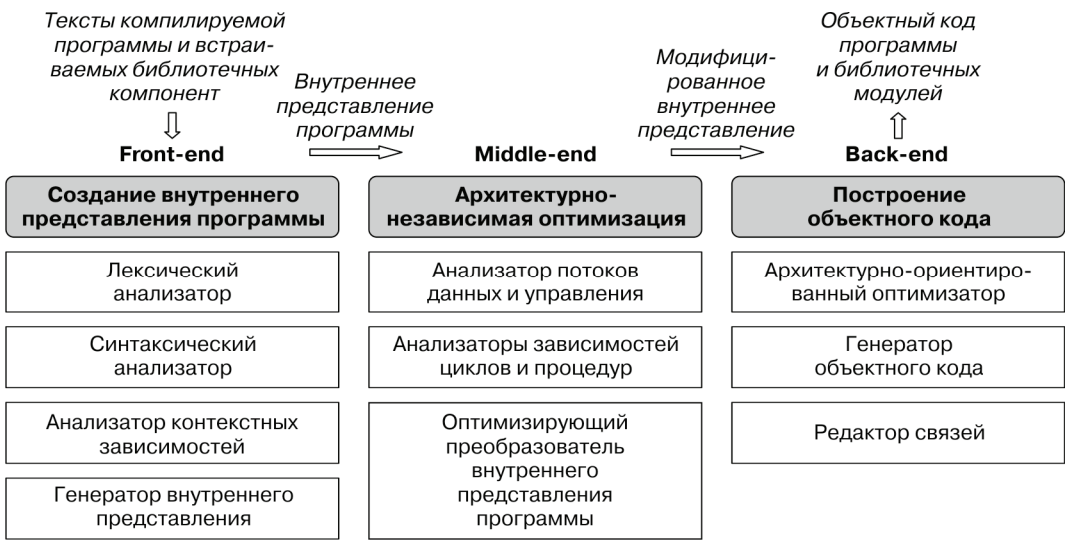

Рис. 1. Структура компилятора

Обсуждая схему компиляции, следует подчеркнуть, что процессы, которые выполняются выделенными компонентами, не обязательно работают последова-

тельно. Единственное условие, связанное с порядком их выполнения, — *зависимость по данным*: процесс, который использует данные, вырабатываемые другим процессом, не может выполняться до того, как эти данные не получены. Пояснение понятия зависимости по данным — хороший повод для рассказа о том, как могут взаимодействовать процессы, если, конечно, обучаемые подготовлены для восприятия этой темы. Если оставаться в рамках проблематики компиляции, то достаточно привести соответствующие примеры, и здесь, имея ввиду предстоящую проблемную задачу, уместно дать представление о процессах, которые реализуются front-end компиляции.

Эта тема включает рассказ о том, что такое *синтаксический анализ* и его задачи, почему в рамках синтаксического анализа выделяется *лексический анализ* как обособленный процесс. Нужно также показать назначение *анализа контекстных зависимостей* и *генерации внутреннего представления* компилируемой программы. Подчеркнем, что совсем не обязательно говорить о методах, реализующих упомянутые процессы, — обучаемым достаточно знать лишь задачи компиляции, решаемые соответствующими компонентами-анализаторами, и то, что некоторые из этих процессов могут выполняться последовательно, один за другим, или совместно, когда активизация одного процесса и его приостановка диктуются зависимостями по данным. Уже здесь прослеживается вариантность реализации компилирующих систем, и учитель может показать преимущества и недостатки вариантов.

Для понимания задачи анализа контекстных зависимостей следует привести несколько примеров. Самый очевидный из них — распознавание типов имен: идентификатор A, описанный как целочисленная переменная, и он же, описанный как переменная вещественная, — это две различные сущности, и, когда компилятор «видит» A, он должен точно знать, о каком имени идет речь в данном контексте компилируемой программы.

Подготовка к формулировке итоговой проблемной задачи требует более подробного разъяснения взаимодействия процессов на примере синтаксического и лексического анализаторов. Синтаксический анализатор (SA) разбирает поток (последовательность) *лексем*, рассматриваемых как неделимые единицы данных; если лексема не соответствует требованиям синтаксиса, то он сообщает об ошибке. Задача лексического анализатора (LA) — генерация лексемы (для любого использования этого модуля безразлично, как это делается): <тип лексемы, атрибуты лексемы>. Откуда и как появляются символы, последовательности которых используются при построении лексемы, совершенно не важно для всех частей компилятора, кроме LA. Он и только он владеет этой информацией, полностью экранируя свое окружение. Отметим, что для реализации лексического анализа недостаточно лишь одной функции генерации очередной лексемы. Этот процесс базируется на задании перерабатываемых данных — текстовых файлов, указателей на них, таблиц идентификаторов, констант и др. совместно с методами работы с этими данными. Следовательно, в любом компиляторе LA — специализированный *модуль*. Это пример, позволяющий не декларативно, а на практике показать преимущества модуляризации программных систем (1).

В качестве примера лексемы удобно привести внутреннее представление идентификатора: тип — имя, атрибуты — последовательность букв и цифр, возможно, иная информация об имени, извлекаемая из его описания или контекста использования. Другой пример, попутно иллюстрирующий возможность влияния SA на LA, — лексемы, которые могут генерироваться из последовательностей символов, подобных «12.4». Если SA распознал, что в текущий момент требуется целочисленное значение, и тем или иным способом сообщил об этом LA, то из указанной последовательности извлекаются «12» и LA генерирует лексему <целое значение, 12>. Оставшиеся символы «.4» будут использованы для формирования лексем при следующих вызовах функции генерации очередной лексемы. Если же SA распознает и сообщает, что требуется вещественное значение, то используется вся последовательность и генерируется лексема <вещественное значение, 12.4> (при условии, что за «12.4» не следует цифра).

Рассказ о Middle-end и back-end для формулировки итоговой задачи не требуется. Тем не менее, для подготовленной аудитории полезно дать обзор процессов, завершающих построение объектного кода, пояснить назначение архитектурно-независимой и архитектурно-ориентированной оптимизации. С точки зрения постановки итоговой задачи достаточно лишь общих сведений об этих процессах, а потому мы в дальнейшем говорим о *семантических подпрограммах*, объединяя этим термином все компоненты компилятора, которые работают с информацией, прямо или косвенно поставляемой синтаксическим анализатором для разных целей: анализ контекстных зависимостей, построение промежуточного представления, его оптимизирующей модификации и др.

Синтаксическое управление. Метод обработки данных, называемый *синтаксическим управлением*, является одним из основных для *сентенциального стиля программирования*, которым разумно пользоваться, когда структура перерабатываемых данных естественно описывается некими формальными правилами и активизацию отдельных действий обработки можно ассоциировать с распознаванием элементов этой структуры — структурных единиц данных [3]. В связи с достаточной общностью метода, применимость которого не ограничивается задачами компиляции, следует остановиться на нем специально. Синтаксическое управление в компиляторах удобно рассматривать лишь как наглядный пример использования метода, а итоговая проблемная задача может служить поводом для разговора об ограниченности адекватного применения вообще любых методов.

Когда говорят о синтаксическом управлении, всегда подразумевается, что *главной подпрограммой* системы объявляется анализатор структуры перерабатываемых данных — синтаксический анализатор, который активизирует другие процессы, выступая в двух качествах. Во-первых, анализатор может *запрашивать информацию*, необходимую ему для продолжения своей работы, а во-вторых, он *предписывает обработку информации*, полученной при анализе, а также накопленной в ходе всей работы системы к текущему моменту. В этом смысле синтаксический анализатор является активным, а все активизируемые им компоненты пассивными элементами действий.

Применительно к компиляции эти общие положения означают, что синтаксический анализатор SA — главная подпрограмма, а лексический анализатор LA и семантические подпрограммы активизируются им. Более подробно взаимодействие компонентов компилятора при синтаксическом управлении описывается следующим образом.

Действия SA как главной подпрограммы синтаксического управления, обусловленные взаимодействием компонентов, сводятся к следующему:

— вызовы подпрограммы из LA, которая формирует лексему и тем самым обеспечивает поставку очередной лексемы по требованию SA и осуществимость продолжения приостановленного синтаксического анализа;

— вызовы семантических подпрограмм, которые обеспечивают выполнение нужных для компиляции действий. После выполнения семантической подпрограммы приостановленный синтаксический анализ продолжается;

— планирование вызов семантических подпрограмм, выполнение которых по тем или иным причинам (чаще всего из-за неготовности перерабатываемой ими информации) *откладывается;*

— передача информации активизируемым подпрограммам.

Комментарий к действиям SA:

— этот компонент ничего не знает о том, как реализуется его требование поставки лексемы от LA, равно как и о том, каким образом активизируемые семантические подпрограммы осуществляют свои действия;

— зависимость SA по данным от LA заключается в том, что не SA может продолжать анализ, если не получит информации об очередной лексеме;

— возможна и обратная зависимость, когда LA не может выяснить, какая из лексем должна порождаться — неоднозначность лексических вариантов. Если синтаксический контекст в состоянии разрешить неоднозначность, то передача информации от SA к активизируемому акту LA исправить ситуацию. Простейший способ разрешения неоднозначности — параметризация вызова подпрограммы из LA, поставляющей лексему (см. обсуждение этого в предыдущем разделе). Другой способ — использование общих глобальных переменных для SA и LA;

— некоторые семантические подпрограммы для своего выполнения требуют информации, которая на текущий момент анализа еще не получена. Это причина, которая обусловливает откладывание обработки данных до момента получения информации.

Действия LA как подчиненной программы, обусловленные взаимным влиянием анализаторов SA и LA, связаны с необходимостью поставки очередной лексемы:

— пассивное ожидание запроса на поставку лексемы, т.е. вызова функции генерации лексемы;

— когда запрос лексемы поступает, LA выясняет, откуда должны выбираться символы для формирования лексемы, т.е. определяет источник последовательности символов. Это может быть текущий источник, т.е. тот файл или иная структура данных, которая была использована при предыдущем формировании лексемы, и тогда следующее действие пропускается;

— изменение текущего источника последовательности символов.

Это действие может быть автономным для LA:

— LA должен вернуться к чтению символов из ранее использованного источника, когда текущий источник исчерпан;

— LA должен переключиться на чтение из нового источника, когда этот источник определяется самим LA (например, при чтении порции очередных символов, обозначающих такой источник при макрогенерации или препроцессировании).

Альтернативный случай — смена источника по предписанию SA. Наиболее простой (но не единственный!) вариант реализации такого предписания — передача нужной информации через параметр функции генерации лексемы;

— когда текущий источник последовательности символов определен, LA осуществляет чтение символов из него и формирование лексемы.

Действия семантических подпрограмм, подчиненных анализатору SA, — это их выполнение, активизируемое SA. Они в целом аналогичны поведению LA:

— пассивное ожидание семантическими подпрограммами вызова;

— некоторая семантическая подпрограмма активизируется для выполнения или для отложенного вызова. Отличие от активизации LA только в том, что, здесь вызывается не одна и та же семантическая подпрограмма, а одна из них. Но, как и при взаимодействии с LA, SA всегда известно, какую именно подпрограмму нужно вызвать или запланировать для отложенного вызова;

— SA влияет на выполнение семантической подпрограммы через параметры или общий контекст точно также, как информация о смене текущего источника последовательности символов передается лексическому анализу;

— после выполнения (неотложенной) семантической подпрограммы управление возвращается SA, и синтаксический анализ продолжается;

— семантическая подпрограмма через параметры или общий контекст может влиять на процесс синтаксического анализа, подсказывая вариант его продолжения. По своей сути такое влияние реализует методы разрешения противоречий анализа, позволяющие работать с недетерминированными грамматиками (в некоторых случаях это может упростить анализ);

— синтаксическое управление подпрограммами, выполнение которых отложено, ограничивается планированием и на их вызовы не распространяется.

Для менее подготовленной аудитории синтаксическое управление можно давать упрощенно, ограничиваясь следующими положениями.

Дисциплина синтаксического управления определяется действиями SA, который:

— активизирует LA и сам решает, как очередная полученная им лексема влияет на его поведение в соответствии с синтаксисом транслируемого языка;

— планирует откладываемые вызовы семантических подпрограмм в соответствии с семантикой языка периода компиляции;

— вызывает семантическую подпрограмму в соответствии с семантикой языка периода компиляции;

— ожидает окончание выполнения вызванных им подпрограмм, после чего возобновляет свою работу.

Рассмотрим метод реализации управления компиляционными процессами, который является альтернативой синтаксическому управлению.

Событийное управление. Стиль программирования, характерной особенностью которого является то, что управление строится как реакция на события, возникающие при функционировании программной системы, называется *событийным*, или *программированием от событий*. Широта его применения и влияние, оказываемое на образ мышления разработчиков систем, сами по себе заслуживают внимания с точки зрения преподавания программирования и информатики. По этой причине мы сначала дадим общее представление о событиях и о том, как они могут управлять вычислениями, а затем покажем возможность применения событийности при разработке компиляторов. Как и ранее, даются лишь общие положения о событийном стиле, а для получения систематической информации читателю рекомендуется обратиться к специальным изданиям [3].

Основа управления событийного стиля — активизация выполнения программных единиц (определенных фрагментов программы: модулей, подпрограмм, и др.) в качестве реакций на распознаваемые события.

В узком смысле событие — это сигнализация окружения программы о том, что на уровне оборудования произошли изменения, требующие произвести определенные действия немедленно. Простейший пример — нажатие клавиши, которое аппаратурой приводится к появлению кода нажатой клавиши в некотором буфере. Если не реагировать на это, т.е. не произвести обработку буфера, то информация о нажатой клавише просто пропадет, например, при следующем нажатии клавиши. Поэтому в системе организуется прерывание выполняемых в текущий момент программ, за время которого программы-реакции отрабатывают необходимые действия.

Метод, возникший при разработке операционных систем, распространяется и на другие области программирования. При этом разумно трактовать события более широко, не ограничиваясь их первоначальным смыслом. Придерживаясь такого расширенного взгляда на событие, мы определяем событие как любое распознаваемое изменение обстановки текущего вычисления (контекст вычисления), требующее специальной обработки, называемой реакцией.

К примеру, можно считать событием наступление момента вычислений, когда значение некоторой переменной становится больше значения другой переменной (неважно, по какой причине), если эта ситуация требует специальной реакции. В этой трактовке распознавание события отделено от обработки. Другой (альтернативный) взгляд на событийность — когда обрабатывающие событие программные единицы «знают» события, на которые они способны реагировать в текущий момент выполнения. В этом случае распознавание есть часть функциональности программных единиц, которые не активизируют действия для «чужих» событий.

С любым событием связана определенная информация, которая называется *сообщением*. Это данные, которые используются при отработки реакции.

На основе введенных понятий событийное управление определяется положениями, фиксирующими то, что происходит при появлении события:

— распознавание — действие, выполнение которого приводит к следующему:
определение программных единиц, которые могут реагировать на событие,

передача всем этим единицам сообщения, связанного с событием, т.е. обеспечение доступа к сообщению;

— реакция — действия, обеспечивающие отработку события:

передача управления выделенным программным единицам; если их несколько, то определяются: последовательное, параллельное или совместное их выполнение, а также, если это необходимо, конкурентное или поочередное выполнение программ обработки. Все это задает дисциплину управления,

выполнение выделенных программных единиц в соответствии с установленной дисциплиной.

В ходе реакции может происходить:

— изменение локальных данных и других атрибутов, относящихся к реагирующей программной единице;

— изменение данных и атрибутов обстановки, т.е. глобальный контекст, включая свойства других программных единиц;

— объявление новых событий, если это требуется для управления.

Для постановки проблемной задачи кроме этих общих сведений нужно дать представление о том, что при организации событийного управления необходима *специальная системная поддержка*, которая обеспечивает распознавание и корректное реагирование. Студенты должны иметь представление о том, что в рамках этой поддержки реализуются действия, связанные с реакциями, и решаются такие вопросы, как организация очередей событий, требующих реакции одной и той же программной единицы, разрешения конфликтов обращения к общим данным и др.

В качестве иллюстрации использования событийного управления далее приводится использование этого метода при организации взаимодействия процессов компиляции. Это альтернативная синтаксическому управлению стратегия. Сопоставление их — суть проблемной задачи, которую мы представим ниже.

Будем рассматривать LA в качестве *генератора событий*. В данном случае генерируется единственное событие, которое в дальнейшем называется *появлением очередной лексемы*. Каждое такое событие сопровождается *сообщением*, содержащим те же самые атрибуты лексемы, о которых говорилось при обсуждении синтаксического управления (см. выше). Синтаксический анализ как процесс, который выявляет структуру компилируемой программы, в зависимости от текущей ситуации должен по-разному реагировать на появление различных лексем. В связи этим определяются состояния SA, в каждом из которых отрабатываются такие ситуации для каждой из лексем, которые в соответствии с синтаксисом языка являются для них допустимыми. Событие с недопустимой лексемой — синтаксическая ошибка компилируемой программы, и для него также должна быть предусмотрена реакция: извещение пользователя о недопустимой ситуации.

Будем называть подпрограммы, связанные с каждой из возможных лексем в некотором состоянии *фрагментами синтаксического анализа*, или просто *фрагментами*. Необходимой функциональностью таких фрагментов является то, что их выполнение всегда заканчивается назначением нового состояния SA. Если новое состояние однозначно определяется допустимой лексемой, с которой связан

фрагмент, то система состояний определена корректно (2). Новое состояние, назначаемое при идентификации ошибки (т.е. когда текущая лексема является недопустимой), связывается со стратегией продолжения анализа в этой ситуации, принятой для компилятора. Простейшее решение — состояние, в котором компиляция прекращается.

Если в текущий момент анализа требуется выполнение или планирование семантической подпрограммы, то это реализуются как часть фрагмента. Наиболее простая для компиляции их активизация — процедурный вызов, завершающийся возвратом к фрагменту. Альтернатива ей — использование событийного механизма, т.е. генерация событий, на которые реагируют семантические подпрограммы, с последующим возобновлением анализа. Эта схема взаимодействия рассматривается при описании действий SA и LA, представленном ниже.

Действия LA, обусловленные событийным управлением, связаны с генерацией события появления очередной лексемы:

— формирование сообщения, связанного с создаваемым событием. Это действие ничем не отличается от того, как формируется лексема при синтаксическом управлении, не исключая автономное изменение текущего источника последовательности символов лексемы. Смена источника по предписанию SA также аналогична, с точностью до того, что здесь нет возможности использовать параметризацию для передачи информации;

— генерация события появления очередной лексемы.

Действия SA, обусловленные событийным управлением, связаны реакцией на событие появления очередной лексемы в текущем состоянии:

— ожидание события, которое порождает LA;

— распознавание в текущем состоянии события появления очередной лексемы. Оно приводит к выявлению реагирующего на это событие фрагмента синтаксического анализа, связанного с лексемой.

Выполнение реакции на появление лексемы, которая сводится к следующему:

— запуск программы выявленного фрагмента, включая его семантических подпрограмм;

— ожидание завершения выполнения фрагмента;

— назначение нового текущего состояния SA;

— в ходе реакции возможна генерация события, на которые реагируют семантические подпрограммы. В частности, для передачи в контекст выполнения LA информацию для влияния на его поведение — аналог параметризации.

Как и в случае синтаксического управления, приведем сводные положения, характеризирующие управление от событий. Для не очень подготовленной аудитории этого достаточно для понимания проблемной задачи, постановка, решение и обсуждение которой рассматривается как цель работы.

Дисциплина событийного управления определяется следующими действиями

— генерация лексем анализатором LA, как относительно автономной (с точностью до влияния на его поведения изменениями контекста) компонентой компилятора;

— последовательность (поток) событий появления лексем (приходящих от LA), которая однозначно определяет фрагмент синтаксического анализа, задающий

реакцию на это событие в текущем состоянии SA в соответствии с синтаксисом компилируемого языка;

— активизация семантических подпрограмм в соответствии с семантикой языка периода компиляции в рамках выполнения фрагмента путем их вызова либо с использованием специальных событий;

— назначение нового текущего состояния SA.

Проблемная задача. Мы рассмотрели, как можно организовывать событийное управление. С помощью событий достигается реализация компиляторных процессов, и событийное управление может рассматриваться как альтернатива синтаксическому управлению. Однако практика показывает, что в реальных компиляторах используется метод синтаксического, а не событийного управления. Из этого следует вопрос, который можно рассматривать как кратную формулировку проблемной задачи: почему обычно реализуют схему синтаксического управления компиляцией, а не событийно ориентированное управление?

Обучаемым предлагаются наводящие вопросы:

— какие преимущества событийного управления вы видите по сравнению с синтаксическим управлением;

— какие преимущества синтаксического управления вы видите по сравнению с событийным управлением;

— когда событийное управление будет предпочтительнее по сравнению с синтаксическим и наоборот;

— когда вы говорите о качестве программной системы, какие критерии качества вы подразумеваете;

— что теряется при выборе одной из двух схем управления по сравнению с альтернативной;

— какой из двух видов управления является, по вашему мнению, более эффективным для компилирующих систем с точек зрения времени компиляции; простоты разработки; понятности программы системы; перспектив развития?

Этот список варьируется в зависимости от аудитории и предпочтений учителя. В любом случае нужно добиться от аудитории осознанного понимания, что никакой метод никогда нельзя считать абсолютно универсальным, что «универсальная система — это плохо специфицированная специализированная система». Для стимулирования активности обучаемых и подготовке их к осознанию уточненной формулировки задачи и восприятию заключительных выводов можно прибегнуть к обсуждению в форме ролевой игры, в которой аудитория разделяется на две антагонистические группы [4]. На каждом шаге игры эти группы принимают роли сторонников конкурирующих методов управления. Группы выступают с защитой своих методов и критикой альтернативных. На следующем шаге группы меняются ролями. Высказанные мнения фиксируются, и после двух-четырех шагов (обычно этого оказывается достаточно) проводится совместное обсуждение доводов. На фоне такого обсуждения удобно вернуться к проблемной задаче, формулируя ее более конкретно:

— какие особенности компиляции делают схему синтаксического управления компиляцией более предпочтительной по сравнению с событийно ориентированным управлением;

— в каких аспектах они проявляют себя: в свойствах языка компилируемых программ, в требованиях к системе программирования и к ее практическому использованию;

— для каких процессов компиляции синтаксическое управление перестает быть предпочтительным (указать критерии предпочтения);

— когда это может приводить к решению об отказе от синтаксического управления?

Трудно ожидать, что начинающие программисты смогут дать ответы, всесторонне и объективно освещающие проблему выбора подхода к компиляции. Новая формулировка задачи является проблемной по своей сути, и, как всегда бывает в таких случаях, она не может быть однозначной. Трансформация ответов на поставленные вопросы в реализационные решения — задача разработки компилирующей системы, которая ставится, когда зафиксированы реальные условия развития проекта. Главная причина постановки проблемной задачи не в стремлении получить рецепт, а в стимулировании у обучаемых потребности думать над подобными вопросами.

Обсуждение поставленной задачи целесообразно завершить наблюдением, касающимся взаимосвязи лексического и синтаксического анализа, которое позволяет показать место событийного механизма как общего метода управления уже безотносительно применения его при разработке компиляторов.

Если проанализировать потоки лексем, которые посредством событий влияют на синтаксический анализ (предопределяют его), и грамматик языков программирования, то становится ясно следующее. Для многих практически значимых языков для всех корректных программ множества лексем, допустимых в каждой ситуации анализа, можно вычислить заранее, указав условия, при которых та или иная лексема обязательно будет представлена в потоке, и действия, которые нужно осуществить для корректного продолжения анализа (3). Это означает, что в рассматриваемом случае все возможные события, управляющие анализом, могут быть вычислены заранее, статически, т.е. до выполнения компилирующей программы. Если это так, то специальный механизм организации ожидания событий и активизации реакций на них может быть заменен простым вызовом процедур без дополнительных накладных расходов.

Изучение синтаксического управления при компиляции в контексте предыдущих рассуждений показывает, что все моменты, когда для продолжения синтаксического анализа требуется, чтобы были выполнены определенные действия, можно оформить как ситуации ожидания подходящих событий, генерируемых соответствующими подпрограммами. Как и в случае событийного управления при компиляции, подобные управляющие события могут быть выполнены статически, получив в точности то управление, которое реализуется исходной схемой. Иными словами, в обоих случаях для компиляторных построений буквальная реализация событийного управления является избыточным.

Компиляторной областью не ограничивается область применимости обсуждаемых методов. Поэтому рассуждения по поводу статического вычисления событийного управления применимы и в других ситуациях. Вывод, который сле-

дует из нашего рассмотрения, сводится к следующему. Во всех случаях, когда можно использовать событийное управление, необходимо проверить возможность его статического вычисления. Если это так, то для принятия решения следует оценить целесообразность его реализации с точки зрения наиболее актуальных в конкретной ситуации критериев: приемлемая эффективность, трудоемкость, сложность развития и др.

Выводы. Предлагаемый подход к преподаванию информатики, связанный с постановкой проблемных задач с неоднозначными решениями, является перспективным для повышения качества обучения программированию. Задача, рассмотренная выше, является всего лишь одним из многих примеров, когда учебную тему можно ставить как проблему, решаемую самими обучаемыми. Это позволяет не просто информировать, но давать сведения о методах контексте, затрагивая смежные вопросы, комплексно представляя изучаемую дисциплину. Главное же в подходе то, что заостряется внимание на границы применимости методов, на многокритериальном выборе метода при реализации программных систем.

Навыки многокритериального выбора метода полезны не только для тех, кто намерен связать свою профессиональную деятельность с программированием и информатикой. Владение этими навыками является критичным для работников тех профессий, которые предполагают принятие решений. По сути дела, любой креативный процесс требует их. Они необходимы для руководителей всех звеньев управления, которым всегда при принятии решений нужно соблюдать баланс интересов работников — исполнителей, заказчиков, контрагентов и субподрядчиков. Особое значение навыки многокритериального выбора имеют для педагогов, которым нужно не только владеть ими, но и обучать этим навыкам своих учеников. И здесь роль преподавателей информатики весьма значительна: это та область творческой деятельности, которая демонстрирует необходимость обоснованного выбора решений в полной мере.

ПРИМЕЧАНИЯ

(1) Автор имел возможность убедиться, что для квалифицированных программистов выделение модулей, подобных лексическому анализу, настолько органично, что они зачастую пропускают его обоснование при построении или объяснении архитектуры системы. Вот характерный случай. Один молодой преподаватель, объясняя тему компиляции начинающим студентам, забыл сообщить, почему лексический анализ реализуется в виде модуля. Ему был дан совет, как исправить ситуацию: повиниться перед студентами на следующем занятии и оправдать себя естественностью такого решения. Дидактическая ошибка была исправлена наилучшим способом: студенты надолго запомнят ситуацию.

(2) Для детерминированных грамматик корректная система состояний может быть построена независимо от того, что выполняется при вычислении фрагментов. В случаях использования недетерминированных грамматик корректность может достигаться, за счет влияния семантических подпрограмм на выбор нового состояния. Таким образом, в ряде случаев однозначность анализа может достигаться точно также, как при синтаксическом управлении.

(3) Это свойство, замеченное Д. Кнутом еще в начале семидесятых годов прошлого века, позволило сформулировать подход к трансляции, названный LR-анализом [5]. Метод LR-анализа прошел путь от идеи до весьма технологичных решений. Сегодня он является базовым в таких популярных инструментов разработки компиляторов как Yacc, Bison и др. [6].

ЛИТЕРАТУРА

[1] *Серебряков В.А.* Лекции по конструированию компиляторов. — М.: ВЦ РАН, 1994.
[2] *Ахо А., Сети Р., Ульман Д.* Компиляторы: принципы, технологии и инструменты. — М.: Вильямс, 2003.
[3] *Непейвода Н.Н., Скопин И.Н.* Основания программирования. — М. — Ижевск: Институт компьютерных исследований, 2003.
[4] *Скопин И.Н.* Ролевые игры в методике обучения руководству проектной деятельностью // Наука и образование. — 2010. — № 1 (57). — С. 74—77.
[5] *Кнут Д.* О переводе (трансляции) языков слева направо // Библиотека Кибернетического Сборника. Языки и автоматы. / Под ред. А.Н. Маслова и Э.Д. Стоцкого. — М.: Мир, 1975. — С. 9—42.
[6] *Yacc & Lex*. URL: http://yacc.solotony.com

PROBLEMATIC TASKS IN STUDYING COMMON METHODS OF COMPUTER SCIENCE AND PROGRAMMING

I.N. Skopin

Programming chair
Novosibirsk state university
Pirogova str., 2, Novosibirsk, Russia, 630090

In work it is offered to consider working out problems, showing various variants of its decisions. Advantages of such approach are shown on an example of a classical problem arising at working out of compiling systems. Two competitive its decisions allow to draw a conclusion that any general method has borders of the adequate application.

Key words: style of programming, compilation, syntactic management, computer science, programming.