УДК 004.43:669:372.8

А. Н. Стась

МЕТОДИКА ОБУЧЕНИЯ РАЗРАБОТКЕ ТРАНСЛЯТОРОВ

Рассмотрена методика обучения технологии разработки трансляторов с языков программирования высокого уровня. Предложенная методика способствует развитию навыков алгоритмического мышления. Особенностью рассматриваемой методики является последовательное изучение основных этапов построения транслятора (лексический анализ, синтаксический анализ, контекстный анализ, интерпретатор обратной польской записи) на примере специализированного учебного языка программирования, содержащего минимальный набор основных алгоритмических конструкций. Разработан интерпретатор с данного учебного языка с использованием алгоритмов на основе конечного автомата, рекурсивного спуска и обратной польской записи.

Ключевые слова: *методика обучения, транслятор, интерпретатор, алгоритмическое мышление.*

В настоящее время, в связи с быстрым развитием информатизации общества, сфера образования начинает претерпевать фундаментальные изменения. Стали пересматриваться содержание и цели, разрабатываться и использоваться новые технологии в образовательном процессе. Все эти изменения приводят к новым требованиям со стороны нанимателей, что приводит к повышенному спросу на квалифицированных сотрудников в области информатики, а особенно в программировании и управлении информационными системами. Кратно возрастают требования к выпускникам вузов. Вполне очевидна алгоритмическая направленность специалиста в области новых информационных технологий.

Проблема общения с компьютерной техникой требует умения понимать различного рода алгоритмические языки, а значит, и наличия определенного уровня сформированности алгоритмического мышления. В течение последних десятилетий произошло плавное движение школьной информатики от технической дисциплины, ориентированной на разработку программ, к дисциплине, направленной на овладение учащимися навыками использования компьютерных технологий в различных сферах человеческой деятельности. Вопросы алгоритмизации и программирования заменяются изучением офисных технологий, которые сводятся в большинстве своем к работе с офисными приложениями. Важнейшая задача формирования стиля мышления и научного мировоззрения у школьников подменяется подготовкой к практической деятельности. На практике эта ситуация приводит к тому, что обучение основам программирования приходится проводить уже в вузе. Чтобы исправить эту ситуацию, необходимо искать качественно новые подходы в обучении программированию студентов педагогических вузов.

Одним из таких подходов может стать введение в учебный план новых дисциплин, задачей которых является не только обучение конкретным технологиям, но и дальнейшее развитие алгоритмического мышления. Одна из таких дисциплин посвящена методам трансляции с языков высокого уровня. Мы можем предположить, что обучение методам трансляции способствует развитию навыков алгоритмической деятельности и алгоритмического мышления.

Целью работы является разработка технологии обучения методам построения трансляторов, способствующей развитию у студентов алгоритмического мышления.

Особенности деятельности программиста, решаемых им задач заставляют говорить об особом типе мышления – алгоритмическом. Понятие алгоритмического мышления рассматривалось в ряде работ по преподаванию информатики и программирования [1]. Не существует какого-нибудь единого и четкого понимания, что же такое алгоритмическое мышление. Один из подходов основан на совокупности мыслительных операций, необходимых в работе программиста.

На наш взгляд, программисту необходимо умение оперировать образами, а также понятиями и категориями, необходимы навыки формирования суждений в области алгоритмизации и программирования.

Безусловно, необходимы навыки индуктивных и дедуктивных умозаключений, способность к обобщению и конкретизации, синтезу. Также достаточно очевидно, что программист должен уметь формализовать стоящую задачу, уметь работать с основными алгоритмическими конструкциями (следование, ветвление, цикл, вызов, косвенная адресация), уметь записывать алгоритмы на специализированных языках [2, 3].

Отдельного рассмотрения требует вопрос о соотношении алгоритмического и творческого в деятельности программиста. В психолого-педагогических исследованиях часто принято противопоставлять алгоритмизм и творчество. Идея о том, что алгоритмизм мешает творчеству, лежит в основе многих инновационных методов обучения – проблемно-развивающее обучение, метод проектов, кейс-метод и другие.

На наш взгляд, с этими оценками мы не можем согласиться, если речь идет об обучении алгоритмизации и программированию. Деятельность программиста, несомненно, является продуктивной и даже творческой. Действительно, программист, обладающий исключительно репродуктивными навыками, сможет лишь повторно решать уже решенные задачи, в лучшем случае переписывать алгоритмы с одного языка программирования на другой. С другой стороны, конечный результат деятельности программиста – это именно алгоритм. И какой бы продуктивной ни была идея, она не будет иметь практического значения в данной области, если не будет представлена в виде четкого алгоритма.

Таким образом, в процессе обучения алгоритмизации и программированию необходимо развивать как репродуктивные, так и продуктивные навыки.

Содержание дисциплины «Трансляция с языков высокого уровня» определено рабочей программой, разработанной автором [4]. Основная идея состоит в том, что последовательно изучаются все этапы построения транслятора – лексический анализ, синтаксический анализ, контекстный анализ, алгоритм интерпретации.

Параллельно, на этапе лексического анализа, углубленно изучаются автоматные грамматики, конечные автоматы и регулярные выражения. На этапе синтаксического анализа – контекстно-свободные грамматики и магазинные автоматы. На этапе алгоритмов интерпретации – обратная польская запись как один из вариантов синтаксически независимого представления программы.

Крайне важным с точки зрения изучения технологии построения трансляторов и с точки зрения развития общих алгоритмических навыков является формирование основных понятий и навыков в области формальных языков. Действительно, для того чтобы эффективно записать программу на том или ином языке, необходимо четко понимать его особенности.

Можно предположить, что изучение формальных описаний основных языков будет способствовать пониманию их внутренних особенностей, а значит, эти знания усилят алгоритмические навыки. В этом отношении примеры из области технологии построения трансляторов являются незаменимыми, они наглядно показывают, что синтаксис большинства языков программирования является контекстно-свободным, и поясняют, что это означает на практике.

Еще одна неотъемлемая составляющая при изучении технологии построения трансляторов – это детальный разбор способа исполнения всех без исключения алгоритмических конструкций и описаний.

Действительно, при изучении алгоритма интерпретации обратной польской записи приходится детально разбирать каждую конструкцию и способ ее выполнения на самом низком уровне. Наиболее показательно в этом плане изучение представлений условных операторов и циклов с помощью комбинаций условных и безусловных переходов, и особенно реализация механизма вызова подпрограмм, включая рекурсию. В процессе реализации алгоритма интерпретации студент вынужден «вручную» прописать все те действия, что выполняет в данный момент компьютер. Эти описания носят даже более детальный характер, чем в программах на ассемблере.

Наш курс построен на основе разработанного учебного языка программирования, с которого был создан транслятор, точнее интерпретатор. В нашем языке все переменные имеют целочисленный тип. Это, с одной стороны, позволит показать пример работы с переменными, но с другой – позволит не запутаться в особенностях работы с каким-то конкретным типом данных. Соответственно, все константы также будут целочисленного типа. Запись идентификаторов стандартна – первый символ должен быть буквой латинского алфавита. Поддерживаются одномерные массивы, но не поддерживаются многомерные. Для доступа к элементу массива используются квадратные скобки. Нумерация индексов начинается с нуля. Этот подход позволяет на наиболее простом из возможных примеров показать принципы внутренней реализации массивов.

В качестве разделителя операторов используем «;». Оператор присваивания – «:=».

Поддерживаются следующие бинарные операции:

+, –, *, / (div), \ (mod), <, >, <=, >=, !=, =

Комментарии заключаются в «|*…….*|».

Перечислим операторы:

INPUT <переменная> – ввод

OUTPUT <выражение> – вывод

IF <выражение> THEN <оператор 1> [ELSE <оператор 2>]

WHILE <выражение> DO <оператор>;

FUNC <название>(<параметр>) – заголовок функции

RET <возвращаемое значение> – возврат из функции

<название функции>(<параметр>) – вызов функции внутри выражения.

HALT – прекращение выполнения программы.

В качестве составного оператора используем фигурные скобки как в языке С.

Итак, имеются простые операторы ввода и вывода, короткая и длинная форма условного опера-

тора. Мы сознательно отказываемся от оператора выбора, поскольку способ его внутренней реализации повторяет реализацию условного оператора, а наша задача сделать пример как можно более компактным. По этой же причине поддерживаем всего один вариант цикла – цикл с предусловием. Также реализована возможность вызова подпрограмм, в том числе поддерживается и рекурсия, но в то же время для простоты ограничимся всего одним параметром.

В процессе лексического анализа параллельно освещаются общие основы математической лингвистики (теории формальных языков), а также способы работы с автоматными (регулярными) грамматиками и конечными автоматами, а в разделе «Синтаксический анализ» изучаются методы работы с контекстно-свободными грамматиками и магазинными автоматами. В качестве синтаксически независимого внутреннего представления используем обратную польскую запись, которая также имеет множество приложений в алгоритмистике и в этом случае более предпочтительна.

Структура предложенной методики представлена на рис. 1.

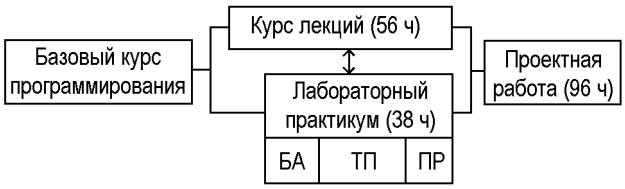

Рис. 1. Методика обучения методам трансляции с языков высокого уровня: БА – базовые алгоритмы; ТП – типовой пример; ПР – проектная работа (установочная часть)

Обратим внимание на то, что курс лекций и лабораторный практикум проводятся параллельно, при этом лекционный курс целесообразно начинать раньше на 18 ч, на которые его длительность превышает длительность лабораторного практикума. Выполнение самостоятельных проектов возможно по окончании как лекционного курса, так и лабораторного практикума.

Лабораторный практикум включает в себя обучение реализации базовых алгоритмов, разбор типового примера интерпретатора на уровне исходных кодов и установочную часть работы студентов над собственными проектами. Эта работа также будет осуществляться в рамках часов, отводимых на самостоятельное обучение.

Примерное тематическое планирование лабораторного практикума следующее:

– изучение алгоритма реализации конечного автомата – 2 ч;

– изучение алгоритма Дейкстры – 4 ч;

– изучение алгоритма оптимизации простой (арифметической) ОПЗ – 2 ч;

– изучение алгоритма интерпретации арифметической ОПЗ – 2 ч;

– изучение алгоритма линеаризации матрицы предшествования – 4 ч;

– изучение типового транслятора – 18 ч;

– установочная часть работы над проектом – 6 ч.

Для более углубленного изучения материала, в рамках самостоятельной работы может быть предложено выполнение собственного проекта на разработку своего транслятора. Это может быть разработка компилятора с этого же языка или разработка транслятора с языка, имеющего другой синтаксис и/или набор команд. Например, можно предложить реализовать циклы с постусловием или с параметром дополнительно или вместо цикла с предусловием. Возможно построение транслятора, поддерживающего несколько типов подпрограмм, или многомерных массивов, или несколько простых типов данных. Возможна также реализация в рамках проектов алгоритмов оптизации внутренних представлений различного уровня.

Занятие можно проводить, следуя методологии проблемного обучения, оно может носить исследовательский характер. В качестве заданий для особо одаренных студентов и/или профессионально заинтересовавшихся изучением данной темы необходимо предложить исследовательские проекты на разработку языков, построенных в соответствии с особыми специализированными парадигмами, для решения узкоспециальных задач. Возможно выполнение проектов на смежные темы, например, направленных на использование синтаксических алгоритмов для задач распознавания образов.

Для оценки алгоритмических навыков до и после изучения спецдисциплины «Трансляция с языков высокого уровня» используем контроль знаний, проводимый на первом и последнем занятиях. Проводится быстрое тестирование (10 заданий) и контрольная работа (4 задания).

В рамках эксперимента было протестировано 12 студентов группы 408 и 8 студентов группы 416.

Средний результат входного контроля в группе 408 – 45,83. Результаты, показанные на выходном контроле, – 61,42. В группе 416 на выходном контроле – 42,5, а на выходном – 51,25. Таким образом, в группе 408 улучшение результатов – 25,37 %, в группе 416 – 20,59 %.

Рассмотрим особенности технологии построения транслятора с созданного учебного языка программирования.

Общая схема построенного интерпретатора приведена на рис. 2.

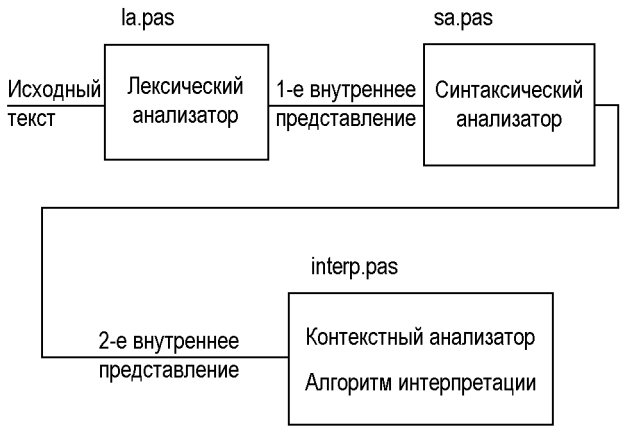

Рис. 2. Структура интерпретатора

Таким образом, интерпретатор состоит из трех модулей. В модуле la.pas реализован лексический анализатор, принимающий на входе исходный текст программы, и генерирующий синтаксически зависимое первое внутреннее представление с выделенными лексемами (токенами). В модуле sa.pas реализуется синтаксический анализатор, который на входе принимает первое внутреннее представление, а на выходе выдает второе внутреннее представление, которое в отличие от первого является синаксически независимым от исходного текста программы. Второе внутреннее представление строим на основе расширенной обратной польской записи. В модуле interp.pas реализованы контекстный анализатор и алгоритм интерпретации обратной польской записи по типу стековой виртуальной машины.

На вход лексического анализатора подаем файл с расширением *.src. В случае если нет лексических ошибок, то лексический анализатор выдаст файл *.ala, в котором хранится первое внутреннее представление, а также файл *.cta, в котором хранится таблица значений констант. Используется следующий формат файла *.cta: в первой строке записано количество констант, в последующих строках – значение каждой из констант. Файл *.ala имеет обычный текстовый формат, но не разделенный на отдельные строки.

На вход синтаксического анализатора подаем файл *.ala. По окончании генерируется файл *.asa (в случае отсутствия синтаксических ошибок), имеющий бинарный формат. На каждый элемент обратной польской записи отводится по 2 байта (1 байт – тип элемента, 1 байт – числовой код). При этом поддерживаются следующие типы элементов: I – идентификатор, C – константа, L – метка, F – функциональная метка, O – операция. Также синтаксический анализатор генерирует файлы *.lab и *.fun, последовательно содержащие смещения, соответствующие меткам и функциональным меткам в бинарном формате (2 байта – на каждое смещение).

На вход интерпретатора фактически подаются файлы *.asa, *.cta, *.lab, *.fun. В случае отсутствия ошибок контекста обеспечивается исполнение программы.

Таким образом, данный интерпретатор можно назвать трехпроходным. Следует обратить внимание на то, что лексический анализатор вызывается отдельным этапом. Выбор такого подхода связан с методической необходимостью обеспечить «прозрачность» работы, т. е. наглядную демонстрацию всего процесса трансляции.

Лексический анализатор строится на основе применения конечного автомата, работающего по схеме, приведенной в таблице.

Схема работы конечного автомата

	a..z, A..Z	0..9	;	п	+,–,/,\,	(	)	[	]	:	=	>,<	!	|	*	{	}
S	A	B	S	S	S	S	S	S	S	C	S	D	E	F	S	S	S
A	A	A	S	I	S	S (ф-ция)	S	S	S	C	S	D	E	F	S	S	S
B	Ош	B	S	S	S	Ош	S	Ош	S	Ош	S	D	E	F	S	Ош	S
C	Ош	Ош	Ош	Ош	Ош	Ош	Ош	Ош	Ош	Ош	S (:=)	Ош	Ош	F	Ош	Ош	Ош
D	A (<,>)	B (<,>)	Ош	S (<, >)	Ош	Ош	Ош	Ош	Ош	Ош	S (<=, >=)	Ош	Ош	F	Ош	Ош	Ош
E	Ош	Ош	Ош	Ош	Ош	Ош	Ош	Ош	Ош	Ош	S (!=)	Ош	Ош	F	Ош	Ош	Ош
F	Ош	Ош	Ош	Ош	Ош	Ош	Ош	Ош	Ош	Ош	Ош	Ош	Ош	Ош	G	Ош	Ош
G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	H	G	G
H	G	G	G	G	G	G	G	G	G	G	G	G	G	Кон ком	G	G	G
I	A (нов)	B (нов)	S	I	S	S (ф-ция)	S	S	S	C	S	D	E	F	S	S	S

Приведем описание синтаксиса нашего языка на уровне первого внутреннего представления в форме Бэкуса–Науэра:

<программа>:–{<оператор>;}*[оператор]

<оператор>:–<составной оператор>|<простой оператор>

<составной оператор>:–{<программа>}

<простой оператор>:–<присваивание>| <ввод>| <вывод>| <условный оператор>| <цикл с предусловием> |<заголовок функции>| <возврат из функции>| <останов>

<присваивание>:–<переменная>:=<выражение>

<ввод>:–K1 <переменная>

<вывод>:–K2 <выражение>

<условный оператор>:–K3 <выражение> K4 <оператор> [K5<оператор>]

<цикл с предусловием>:–K6 <выражение> K7 <оператор>

<заголовок функции>:–K8 <имя функции> (<имя переменной>)

<возврат из функции>:–K9 <выражение>

<останов>:–K10

<выражение>:–<операнд>| (<выражение>) | <выражение><операция><выражение>

<операнд>:–<переменная>| <константа>| <вызов функции>

<операция>:– +| –| *| /| \| <| >| <=|>=| =| !=

<переменная>:– <имя переменной>| <имя переменной>[<выражение>]

<вызов функции>:– <имя функции> (<выражение>).

Лишние пробелы, комментарии, переводы строки и т. д. удалены на этапе лексического анализа.

От описания в форме Бэкуса–Науэра легко перейти к описанию в виде контекстно-свободной грамматики с учетом представления на первом внутреннем языке.

Таким образом, мы получаем формальную грамматику, в соответствии с которой будем анализировать синтаксис. Применяем метод рекурсивного спуска, который основан на моделировании LL(k) – анализа без необходимости полного моделирования магазинного автомата, предварительно устранив в исходной грамматике прямую и косвенную левую рекурсию.

Иногда для установления корректности исходного текста программы недостаточно лексического анализа и синтаксического анализа, и в этом случае задачей контекстного анализа является установление свойств объектов и их использования. Наиболее часто решаемой задачей является определение существования объекта и соответствия его использования контексту, что осуществляется с помощью анализа типа объекта. Под контекстом здесь понимается вся совокупность свойств текущей точки программы, например множество доступных объектов, типы выражений и т. д.

Для решения задачи контекстного анализа существует несколько подходов, из которых выбирается оптимальный для данного языка программирования. Поскольку в рассматриваемом языке определения областей действия переменных отсутствуют, то контекстный анализ будет сводиться к инициализации таблиц идентификаторов и констант, а также к разделению идентификаторов на простые переменные и массивы, которые связаны с необходимостью применения разных принципов при организации хранения данных в процессе интерпретации.

На этапе интерпретации используется стандартный алгоритм для расширенной обратной польской записи.

Таким образом, в процессе работы решены следующие задачи:

1. Исследованы способы развития алгоритмического мышления в процессе обучения технологии построения трансляторов.

2. Разработано содержание дисциплины «Трансляция с языков высокого уровня.

3. Разработан синтаксис специализированного учебного языка программирования, предназначенного для обучения построению трансляторов.

4. Разработан интерпретатор с учебного алгоритмического языка, включающий лексический анализатор, синтаксический анализатор и интерпретатор обратной польской записи.

Список литературы

1. Психология мышления. URL: http://libsib.ru/obschaya-psichologiya/psichologiya-mishleniya (дата обращения: 18.11.2013).
2. Стась А. Н., Долганова Н. Ф. Развитие алгоритмического мышления в процессе обучения будущих учителей информатики // Вестн. Томского гос. пед. ун-та (TSPU Bulletin). 2012. Вып. 7 (122). С. 241–244.
3. Якименко О. В., Стась А. Н. Применение обучающих программ-тренажеров в обучении программированию // Вестн. Томского гос. пед. ун-та (TSPU Bulletin). 2009. Вып. 1 (79). С. 54–56.
4. Рабочие программы дисциплины «Трансляция с языков высокого уровня». URL: http://tspu.edu.ru/images/fmf_news/UMKD/230400.62_Informacionnye_sistemy_i_tehnologii/B_3_V_07__Translyaciya_s_yazikov_visokogo_yrovnya.doc (дата обращения: 05.04.2013).

Стась А. Н., зав. кафедрой, ведущий инженер-программист.
Томский государственный педагогический университет.
Ул. Киевская, 60, Томск, Россия, 634061.
E-mail: stasandr@tspu.edu.ru

Материал поступил в редакцию 15.02.2015.

A. N. Stas

METHODS OF TEACHING THE DESIGNING OF TRANSLATORS

This article presents the methods of teaching technology of the development of translators from high-level programming languages. The proposed methods contributes to the development of algorithmic thinking skills. The peculiarity of the considered methods is a consistent study of the main stages of building a compiler such as lexical analysis, syntactic analysis, semantic analysis, reverse polish notation interpreter. The study is based on the example of a specialized educational programming language that contains the minimum set of basic algorithmic structures. The interpreter from this language with the use of algorithms based on finite state machine, recursive descent parser and reverse polish notation was developed.

Key words: *methods of teaching, translator, interpreter, algorithmic thinking.*

References

1. *Psikhologiya myshleniya* [Psychology of thinking]. URL: http://libsib.ru/obschaya-psichologiya/psichologiya-mishleniya (accessed: 18.11.2013) (in Russian).
2. Stas A. N., Dolganova N. F. Razvitiye algoritmicheskogo myshleniya v protsesse obucheniya budushchikh uchiteley informatiky [Algorithmic thinking development when training computer science teachers]. *Vestnik Tomskogo gosudarstvennogo pedagogicheskogo universiteta – TSPU Bulletin*, 2012, vol. 7(122), pp. 241–244 (in Russian).
3. Yakimenko O. V., Stas A. N. Primeneniye obuchayushchikh program-trenazherov v obuchenii programmirovaniyu [Use of Computer Tutors in Teaching Programming]. *Vestnik Tomskogo gosudarstvennogo pedagogicheskogo universiteta – TSPU Bulletin*, 2009, vol. 1(79), pp. 54–56 (in Russian).
4. *Rabochye programmy distsipliny "Translyatsiya s yazykov vysokogo urovnya"* [The working program of the discipline "Transmission with high-level languages"]. URL: http://tspu.edu.ru/images/fmf_news/UMKD/230400.62_Informacionnye_sistemy_i_tehnologii/B_3_V_07__Translyaciya_s_yazikov_visokogo_yrovnya.doc (accessed: 05.04.2013) (in Russian).

Stas A. N.
Tomsk State Pedagogical University.
Ul. Kievskaya 60, Tomsk, Russia, 634061.
E-mail: stasandr@tspu.edu.ru