ПРОГРАММИРОВАНИЕ В СТАРШИХ КЛАССАХ ШКОЛЫ И В ВУЗЕ

Е. В. Касьянова, С. Н. Касьянова*

Институт систем информатики им. А. П. Ершова СО РАН,
630090, Новосибирск, Россия,
Новосибирский государственный университет,
630090, Новосибирск, Россия,
*Институт автоматики и электрометрии СО РАН,
630090, Новосибирск, Россия

УДК 004.4:[378.147+373.5]

В статье авторы рассматривают свой многолетний опыт преподавания программирования в школе № 130 им. академика М. А. Лаврентьева и проведения практикума по программированию для студентов механико-математического факультета Новосибирского государственного университета.

Ключевые слова: программирование, алгоритмы, язык программирования, практика конструирования алгоритмов, опыт преподавания, технология преподавания.

In this paper the authors consider their long-term experience of teaching programming at Academician M. A. Lavrentiev school N 130 and experience of conduct of practical work on programming for students of Mechanics and Mathematics Department of Novosibirsk State University.

Key words: programming, algorithms, programming language, algorithm construction practice, teaching experience, teaching technique.

Мы уже много лет преподаем программирование в старших классах лицея № 130 имени академика М. А. Лаврентьева и в Новосибирском государственном университете (НГУ). В нашей школе в 9, 10 и 11 классах с математическим уклоном на занятия по программированию сейчас отводится лишь один урок (40 мин.) в неделю. В компьютерном классе для занятий поставлена система программирования Free Pascal.

Сейчас, когда во многих высших учебных заведениях студенты программируют на языке C++, мы решили провести эксперимент — попробовать со школьниками изучать этот язык. Для этого выбрали в классе самого сильного ученика и предложили ему изучать C++ и выполнять все задания на этом языке программирования, когда весь класс будет работать на языке Паскаль. Несколько лет назад у нас уже была попытка работы на C++ со всем классом, она провалилась: весь класс не осилил этот язык, и мы вернулись к Паскалю. Язык C++ не является простым даже для профессионалов. Достоинством этого языка является то, что программы, написанные на C++, можно переносить на любую платформу, под любой компилятор ANSI (Американский национальный институт стандартов) C++. Хотя создан он был для использования в операционной системе UNIX, сейчас C++ перенесен на многие другие операционные системы. Он является самым популярным языком для создания системного программного обеспечения. Его также часто используют для создания прикладных программ. Но язык C++ разрабатывался не для

новичков. А главное — по силам ли он школьникам? Мы решили проверить это на одном ученике.

Так как был выбран сильный мальчик, то он справлялся с заданиями, но ему было сложнее, чем остальным. Иногда при очередном зависании системы он не выдерживал и переходил на Паскаль. Но через какое-то время мы опять начинали работать на C++. Средний, а тем более слабый школьник не выдержал бы такого испытания. Работать на C++ начинающим программистам сложнее по следующим причинам:

1) в системе программирования Microsoft Visual даже для небольшой программы, написанной на C++, надо создавать проект;

2) система Microsoft Visual загружается намного медленнее, чем Free Pascal;

3) программа, написанная на C++, компилируется в несколько раз медленнее, чем программа, написанная на Паскале;

4) система программирования Microsoft Visual „зависает“ чаще, чем система программирования Free Pascal.

А плюс изучения C++ в школе всего один — подготовка к обучению в высшем учебном заведении. Это, конечно, очень важно, но таким образом можно кого-то хорошо подготовить к дальнейшему обучению, а у кого-то отбить всякий интерес к программированию.

Учитывая все минусы работы школьника в системе программирования Microsoft Visual, мы пришли к выводу, что среднему ученику за короткое время не справиться с такой работой. Не зря одной из главных целей при создании языка Паскаль было построение прочных основ обучения принципам программирования. Язык программирования Паскаль был разработан швейцарским ученым Никласом Виртом в 1968–1970 годах для обучения студентов программированию и получил широкое распространение благодаря наглядности программ и легкости при изучении. По мнению Н. Вирта, язык должен способствовать дисциплинированному программированию, поэтому, наряду со строгой типизацией, в Паскале сведены к минимуму возможные синтаксические неоднозначности, а сам синтаксис автор постарался сделать интуитивно понятным даже при первом знакомстве с языком.

В 1992 г. американская корпорация Borland International выпустила два пакета на языке Паскаль — это Borland Pascal 7.0 и Turbo Pascal 7.0. Эти пакеты используют новейшие достижения в программировании. Язык этих версий обладает широкими возможностями, имеет большую библиотеку модулей, очень удобна среда программирования. Язык Паскаль стараниями идеолога американской корпорации Borland Андерса Хейлсберга превратился в мощную современную профессиональную систему программирования, которая позволяет создавать как простые программы, предназначенные для решения несложных задач, так и разрабатывать сложные реляционные системы управления базами данных. Важным шагом в развитии языка является появление свободных реализаций языка Паскаль — Free Pascal и GNU Pascal, которые обеспечили чрезвычайно широкую переносимость написанных на нем программ на различные платформы под различные операционные системы. В настоящее время создана реализация языка для платформы .Net.

В школе очень важна легкость изучения языка. Именно легкостью языка можно завлечь, а затем и увлечь школьника заниматься программированием. Простота Паскаля позволяет быстро его освоить и создавать алгоритмически сложные программы. В языке реализованы идеи структурного программирования, что делает программу наглядной, а развитые средства представления структур данных обеспечивают удобство работы как с числовой, так и с символьной информацией. В Паскале появились строки символов пе-

ременной длины, открытые массивы, давшие возможность использовать одни и те же процедуры для обработки одномерных массивов различных размеров.

Поэтому мы после эксперимента с языком C++ решили, что в нашей школе, где количество часов, отведенных на изучение программирования, мало́, надо изучать Паскаль, а C++ можно давать на спецкурсе для сильных учеников, которые уже знакомы с Паскалем. Или другой вариант — делить класс на две части. С сильной половиной изучать C++, а с остальными ребятами работать на Паскале. Пока у нас класс разделен на две части по другому признаку. Одна половина — это ребята, которые сдают ЕГЭ по информатике, вторая — не сдают. Это тоже приходится учитывать, а выбирают ЕГЭ по информатике далеко не самые сильные программисты. Поэтому заниматься программированием на C++ ни в одной половине класса, ни в другой не получается.

Так как мы обе являемся преподавателями НГУ, то, наблюдая за студентами первого курса, можем сказать, что студенты, которые со школы хорошо умеют программировать на языке Паскаль, легко переходят на C++. Трудно приходится тем студентам, которые в школе вообще не использовали никакого языка программирования, либо не имели практики программирования, а просто изучали конструкции языка.

Поэтому выделенные для программирования в школе 102 часа мы стараемся использовать с наибольшей пользой. В конце обучения мы отводим время под выполнение самостоятельных проектов. На уроках ребята решают задачи из учебников [1, 2, 3], но, как правило, вся группа пишет одни и те же программы, за исключением одного или двух человек, которые решают задачи сложнее. И если ученик не может отладить программу, то он всегда может взять ее у соседа по сети. А проекты не позаимствуешь, они все индивидуальные. Приходится потрудиться самому. Разрешается консультироваться у учителя, у соседа, но писать и отлаживать программы все равно надо самому.

При выполнении проекта используются средства динамической компьютерной графики, что делает процесс обучения увлекательным и позволяет визуально проверить работу программ. Задачи даются не очень сложные. Например, вывести на экран меню, состоящее из трех строк. Затем движением стрелок можно выбрать одно из трех заданий и запустить на выполнение. Заданием может быть вывод графика заданной функции, движение объекта по заданной траектории на экране, движение объекта под управлением стрелок и т. д. Такая задача для ученика 10 класса вполне по силам.

Наш многолетний опыт показывает, что самостоятельные практические занятия очень полезны и для изучения программирования, и для закрепления полученных знаний.

В курсе программирования для школьников большое внимание уделяется изучению и составлению алгоритмов. Преподавание алгоритмизации учит гибкости мышления и закладывает важную базу для дальнейшего изучения программирования. В технологии преподавания в основном используется проблемный („задачный“) подход в обучении. Почему выбран этот подход? Учитывается психология одаренных детей. Творческие дети как к кроссвордам относятся к решению задач, к „разгадыванию“ алгоритмов. Такие дети не любят решать однотипные задачи, поэтому здесь очень важен подбор задач. Им нужны интересные задачи, при решении которых они приобретают новые знания, знакомятся с новыми методами и новыми структурами данных. Набор задач должен с одной стороны предоставлять возможность ребенку проявлять смекалку, с другой стороны требовать определенных знаний методов решения задач. И задача учителя — превратить обучение в серию пусть небольших, но приятных побед.

За время работы с одаренными детьми нами были собраны задачи повышенной сложности и подготовлен сборник задач с решениями [1]. Речь идет об углубленной подготовке школьников по программированию с учетом их индивидуальных способностей и увлечений. При этом обучение в школе не дублирует университетские курсы по программированию для математиков, а лишь подводит школьников к вузовским программам.

При обучении программированию в вузе наиболее важным, на наш взгляд, является начальный этап, на котором обучаемый должен овладеть навыками точного формулирования алгоритмов на языке высокого уровня, что невозможно сделать, прочитав лишь несколько руководств или прослушав курс лекций по программированию. Необходима практика конструирования алгоритмов, и здесь не обойтись без подходящего набора примеров и задач.

Поэтому неслучайно уже много лет на механико-математическом факультете НГУ проводится практикум по программированию для студентов первого курса. Основной задачей практикума является не только и не столько обучение студентов собственно записи (кодированию) известного алгоритма на языке программирования высокого уровня, а практическое закрепление знаний, получаемых в курсе по программированию, и овладение общими методами, приемами и навыками технологии решения задач на компьютере. Тематика заданий общего практикума определяется не столько конкретными областями знаний, из которых должны браться задачи для их решения на компьютере (это имеет место в специализированных практикумах на старших курсах), а всеми видами работ, которые должен освоить студент, чтобы научиться создавать качественные (эффективные, наглядные и надежные) программы.

В большинстве случаев задача, решаемая во время общего практикума, — это задача невычислительного характера, требующая разработки алгоритма и связанная с обработкой сложных структур данных. Как правило, решаемая задача имеет краткую и точную (содержательную) формулировки и допускает большое разнообразие решений, из которых студент должен выбирать по возможности лучшее.

В 2013 году было издано учебное пособие „Практикум по программированию“ [4]. Пособие предназначено для использования студентами и преподавателями при организации общего практикума, подкрепляющего основной курс по программированию. Книга базируется на богатом опыте преподавания программирования на механико-математическом факультете НГУ [2, 5, 6] и отражает состояние практикума на 2012 год.

Практикум проходит для студентов первого курса во втором семестре одновременно с основным курсом по программированию и на данный момент ориентируется на использование языка C++ и систем Visual Studio и Borland Developer Studio. Это не значит, конечно, что для решения заданий общего практикума, представленных в данной книге, не подходят другие реализации языка C++ или другие языки программирования.

Книга содержит 500 индивидуальных заданий различной сложности, ориентированных на приобретение студентами навыка практического решения задач, требующих разработки алгоритма, обработки сложных структур данных и создания дружественного интерфейса. Основную часть книги составляют подробные рекомендации по выполнению разных видов работ, которые должен освоить студент, чтобы научиться создавать эффективные, наглядные и надежные программы решения таких задач.

Каждое индивидуальное задание — это самостоятельная, как правило, комбинаторная или логическая задача с краткой и четкой формулировкой, не содержащей описания алгоритма.

Вместе с тем студенту нужно иметь в виду, что многие из методов и понятий, используемых в заданиях общего практикума, являются неотъемлемой частью образования современного специалиста, использующего компьютер для решения своих задач. Поэтому студенту рекомендуется прочитать весь методический материал, включая словарь понятий (используемых в заданиях), прежде чем приступать к решению заданий.

Тематические задачи разбиты на пять разделов по 100 заданий в каждом разделе: графы и системы дорог; грамматики, языки и автоматы; формулы и программы; геометрия; игры и модели.

В ходе практикума каждый студент решает пять индивидуальных задач, по одной из каждого раздела. Выполнение задания включает следующие виды работ:

1) анализ условия задачи и выработку подхода к ее решению;

2) пошаговую разработку (на основании выбранного подхода) алгоритма решения и его описание;

3) обоснование алгоритма;

4) выбор и обоснование представления для входных, выходных и промежуточных данных;

5) кодирование алгоритма, т. е. его запись на языке C++;

6) выбор и обоснование набора тестов, на которых будет проверяться программа;

7) отладку программы и демонстрацию ее правильной работы на выбранном наборе тестов.

Это разбиение условно в том смысле, что фактически некоторые виды работ тесно переплетаются и выполнение их обычно составляет единый процесс. Например, строить набор тестов удобнее одновременно с построением самого алгоритма, а обосновывать правильность работы алгоритма удобно путем детальной демонстрации процесса его построения. Практические рекомендации по выполнению всех перечисленных работ приведены в учебном пособии.

В процессе решения задания студент составляет отчет (как правило, в электронном виде), который должен включать:

1) формулировку задачи;

2) описание программы для пользователя, ее внешнюю спецификацию, т. е. описание способа задания входных данных, вида результатов программы при заданных входных данных и сценария диалога в процессе исполнения программы;

3) словесное описание алгоритма и обоснование его правильности и эффективности;

4) текст программы;

5) описание тестового набора и его обоснование;

6) сведения об отладке.

Пример студенческого отчета содержится в учебном пособии.

Важно, чтобы студент уже в начале практикума знал свои задания и все требования, предъявляемые преподавателем к их выполнению, в том числе виды работ и контрольные точки по каждому заданию, порядок и сроки прохождения контрольных точек и сдачи заданий. Цель контрольных точек — постоянный контроль текущего состояния решения заданий для координации установленных преподавателем требований и действительных намерений студента. В любом случае, понимание условий задач студентом не должно отличаться от понимания преподавателем. Поэтому раньше, чем студент начнет разрабатывать алгоритм, студент и преподаватель должны вместе тщательно проанализировать условие задачи и обсудить особенности и трудности ее решения, зафиксировать интерфейс

программы, включая представление входных и выходных данных. Результатом этого согласования должны стать разделы 1 и 2 студенческого отчета.

Необходимо уделять особое внимание высокой надежности создаваемых программ. Программа должна содержать достаточное число так называемых стопоров ошибок — динамических проверок справедливости утверждений, характеризующих правильность функционирования программы и ее применения. Очень важно, чтобы программа не только давала правильные результаты для корректных исходных данных, но и осмысленно реагировала на некорректные данные и указания пользователя. Не менее важно, чтобы программа выдавала на экран (или в специальный файл) в удобном виде информацию о ходе вычисления по программе в таком объеме (и в таком виде), который позволяет легко идентифицировать ошибку (как в программе, так и входных данных) и локализовать место ее возникновения.

Программа обязательно должна быть наглядной, т. е. хорошо структурированной, с достаточным количеством комментариев и т. д., а также содержать инструкцию по своему использованию („встроенный help“).

Особое внимание нужно уделить правильности программы и полноте тестового набора. Описание каждого теста, помимо файла с входными данными, должно включать информацию, достаточную для оценки правильности работы программы на этих входных данных. Минимальное требование к тестовому набору состоит в том, что каждый оператор программы должен быть достижим (т. е. выполняться) хотя бы на одном тесте из этого набора. Поэтому среди тестов набора должны быть представлены и некорректные исходные данные.

Представления как входных, так и выходных данных разработанной программы должны быть естественными для человека. Степень естественности представления определяется объемом той работы, которая требуется для перехода от содержательного описания входных данных к их представлению (очень часто неестественность входного представления проявляется в его неоправданно большом размере) и от представления выходных данных к их содержательному пониманию.

Задачи, составляющие индивидуальные задания, допускают, как правило, целый спектр различных по эффективности решений. Предполагается, что студент должен выбрать и обосновать по возможности хорошее решение. Минимальным требованием к эффективности решения является успешная работа программы на согласованном с преподавателем тестовом наборе.

Для удобства все задачи разбиты на три группы: обычной сложности (их формулировки не имеют пометки), повышенной сложности (с пометкой $\uparrow$) и пониженной сложности (с пометкой $\downarrow$).

При оценке сложности задачи рассматривались три показателя: сложность структур данных, сложность вычислений и изобретательность.

В показатель „изобретательность“ включались такие свойства задания, как непривычность для студента понятий, используемых в задании; сложность извлечения из определений тех свойств, на которых должен базироваться алгоритм решения задания; сложная связь между структурами данных и вычислениями.

Каждый из указанных трех показателей задачи оценивался в баллах 0, 1 или 2. Пометку $\downarrow$ получили задачи, набравшие в сумме по трем показателям всего 1 балл, а пометку $\uparrow$ — задачи, суммарный балл которых больше 3.

Предполагается, что студент, решая задачу, создает интерактивную программу, которая за один запуск может обрабатывать не отдельный входной набор, а последовательность входных наборов произвольной длины. Эти входные наборы либо заранее размещаются в специальных „входных“ файлах программы, предъявляемых преподавателю вместе с разработанной программой, либо задаются пользователем программы в процессе ее исполнения. Таким образом, разработанная студентом программа после завершения обработки очередного входного набора в зависимости от указания пользователя может либо завершить свою работу (остановиться), либо продолжить ее и перейти к обработке следующего входного набора. В последнем случае программа может вступить в диалог с пользователем, в процессе которого пользователь программы может подготовить и инициировать новый счет по программе. Для этого он может либо осуществить весь очередной ввод с помощью клавиатуры и мышки, либо дать указание программе, из какого файла ей нужно взять необходимые данные. В случае диалогового характера разработанной программы студент должен описать сценарий работы с программой, демонстрирующей корректность ее функционирования на тестовом наборе. Этот сценарий является необходимой частью отчета, связанной с выбором и обоснованием набора тестов, на которых будет проверяться программа.

При этом, создавая программу обработки последовательности наборов входных данных, совсем не требуется создавать программу так, чтобы при ее переходе к обработке следующего входного набора этот набор каждый раз целиком задавался заново. В частности, во многих заданиях удобно разделить входной набор на две части, выделяя более общие (как правило, более объемные) исходные данные задачи в отдельную часть, и предусмотреть специальный вид реализации счета по программе на последовательности входных данных набора, различающихся второй частью, без необходимости повторного задания первой.

Например, в задаче нахождения кратчайшего пути между двумя заданными вершинами заданной системы дорог такой более общей частью входного набора является система дорог. Поэтому естественно решение, при котором система дорог является общей для нескольких следующих друг за другом счетов по программе. В этом случае каждый раз, переходя к новому счету по программе, пользователь может выбрать один из двух вариантов продолжения работы:

1) осуществить ввод очередной системы дорог из заданного файла с выводом ее изображения на экран;

2) запустить нахождение кратчайшего пути между двумя заданными городами в текущей системе дорог.

Список литературы

1. Касьянова С. Н., Касьянова Е. В. Программирование для школьников: сборник задач повышенной сложности с решениями. Новосибирск, 2002. (Препринт / ИСИ СО РАН № 95).

2. Касьянов В. Н. Курс программирования на Паскале в заданиях и упражнениях. Новосибирск: НГУ, 2001.

3. Окулов С. М., Ашихмина Т. В., Бушмелева Н. А. и др. Задачи по программированию. Москва: Бином. Лаборатория знаний, 2006.

4. Касьянов В. Н., Касьянова Е. В. Практикум по программированию. Новосибирск: НГУ, 2013.

5. Касьянов В. Н., Сабельфельд В. К. Сборник заданий по практикуму на ЭВМ. М.: Наука, 1986.

6. Касьянов В. Н. Сабельфельд В. К., Трахтенброт М. Б. Сборник индивидуальных заданий по практике на ЭВМ. Новосибирск: НГУ, 1981.

Касьянова Елена Викторовна — канд. физ.-мат. наук, доц., старш. науч. сотр. Института систем информатики им. А. П. Ершова СО РАН, доц. Новосибирского государственного университета; e-mail: kev@iis.nsk.su

Касьянова Светлана Николаевна — ведущ. инженер-программист Института автоматики и электрометрии СО РАН, науч. сотр. Института систем информатики им. А. П. Ершова СО РАН, старш. преп. Новосибирского государственного университета; e-mail: nova@iae.nsk.su

Дата поступления — 20.04.2014