РАЗДЕЛ III
СОДЕРЖАНИЕ И ТЕХНОЛОГИИ ПРЕПОДАВАНИЯ КОМПЬЮТЕРНЫХ ДИСЦИПЛИН В УЧРЕЖДЕНИЯХ ВЫСШЕГО ОБРАЗОВАНИЯ

УДК 51 (075.8)

НЕКОТОРЫЕ МЕТОДИЧЕСКИЕ АСПЕКТЫ ПРЕПОДАВАНИЯ МЕТОДОВ ПРОГРАММИРОВАНИЯ

Н. А. Аленский[1)], **Д. В. Филимонов**[2)]

[1)] *Белорусский государственный университет, пр. Независимости, 4, 2203030, Беларусь, email: alensky@bsu.by*
[2)] *Белорусский государственный университет, пр. Независимости, 4, 2203030, Беларусь, email: dzfilimonau@gmail.com*

Изложена позиция авторов по вопросу совершенствования преподавания методов программирования на первом курсе механико-математического факультета БГУ в связи с развитием информационных технологий и их использованием в образовательном процессе.

Ключевые слова: программирование; методика преподавания; лекция.

SOME METHODOLOGICAL ASPECTS TEACHING PROGRAMMING METHODS

N. A. Alensky[1)], **D. V. Filimonov**[2)]

[1)] *Belarusian State University, Niezavisimosti pr., Belarus, 2203030, email: alensky@bsu.by*
[2)] *Belarusian State University, Niezavisimosti pr., Belarus, 2203030, email: dzfilimonau@gmail.com*

The position of the authors on the problem of improving the teaching of programming methods in the first year of the Faculty of Mechanics and Mathematics of the Belarusian State University in connection with the development of information technologies and their use in the educational process is stated.

Keywords: programming; teaching methods; lecture.

Введение

Основные **цели** дисциплины «Методы программирования»: развить алгоритмическое мышление, изучить современные структуры данных, методы и технологии программирования и с их помощью научиться решать задачи математики, механики, а также информационно-поисковые, текстовые и другие задачи.

Важным в постановке названной дисциплины является **выбор системы программирования,** которой для всех специализаций, кроме будущих педагогов, с первого семестра на ММФ является *C++ Visual Studio.* Более того, на основе многолетнего преподавания этого языка программирования не только для студентов, но и для школьников сделан вывод, что при разумном выборе порядка рассмотрения тем и грамотной методики преподавания C++ можно успешно использовать и в средних общеобразовательных учебных заведениях. Это подтверждается использованием данного языка при изучении информатики в десятых и одиннадцатых классах средних общеобразовательных учебных заведений на повышенном уровне (три часа в неделю вместо одного).

Лекции

Основная идея, связанная с **содержанием** «живых» **лекций**, заключается в следующем: не только во время практических занятий надо не просто описывать язык программирования, а с его помощью учить разрабатывать качественные программы, что не одно и то же. Основное внимание необходимо уделить логически сложным и важным классическим вопросам, которые вызывают обычно затруднения и в меньшей степени зависят от версий системы программирования. Уже на лекции, а не только на практических занятиях, надо развивать алгоритмическое мышление студентов. Логически простые темы с громоздкими и сложными синтаксическими правилами, в которых, кроме этого, много справочного материала, можно предложить изучить самостоятельно, кратко прокомментировав соответствующие электронные материалы с помощью технических средств. Сэкономленное время можно использовать для более подробного объяснения вопросов, которые студенты усваивают с трудом.

Наиболее сложные и объемные темы желательно начинать изучать с разработки программ или их фрагментов, разбора упражнений и тестов во время лекции. На этом этапе рассматриваются параллельно и некоторые теоретические вопросы. После этого студентам предлагается самостоятельно найти ответы на другие заранее подготовленные вопросы, упражнения и тесты, которые размещены, конечно, в образовательном портале

факультета. При этом, используя дифференцированный подход, можно пометить как простые, так и наиболее сложные, но обязательные для всех студентов вопросы; дополнительные вопросы для желающих; вопросы на 9-10 баллов. На последующих лекциях, связанных с данной, вместо классического монолога лектора предлагается использовать диалог, беседу со студентами, уделив основное внимание тому, что вызвало наибольшие затруднения при самостоятельном изучении материала.

Электронные материалы вносят свои коррективы в методику преподавания этой дисциплины. «Живая» лекция не должна полностью дублировать электронную. Желательно по возможности для объяснения нового материала или для его закрепления использовать задачи и упражнения, отличные от приведенных в электронном варианте. Эти положения подкрепляются конкретными примерами.

Такие электронные средства как **проектор** или **интерактивная доска**, эффективны, прежде всего, при изучении простых вопросов, когда надо не столько объяснять, сколько показывать, например, визуальное программирование. Они также помогают быстрее понять громоздкую с точки зрения текста программу, расположив весь код перед глазами; примерами таких тем являются структуры и классы. Но при изучении тем, в которых важен сам процесс разработки алгоритма и программы со сложной логикой, лучше отказаться от разбора готовых решений. Вместо этого предлагается реализовать логику с активным участием студентов и используя технические средства, чтобы результат их предложений был виден в действии. При этом важно, что обучаемый имеет возможность следить, как преподаватель думает во время написания программы.

К изложенной выше методике могут возникнуть и некоторые **вопросы,** ответы на которые не всегда однозначны. Когда «живая» лекция может полностью дублировать электронный материал? Как решить проблему конспектирования «живых» лекций? Оценивать ли во время беседы на лекции ответы на вопросы после самостоятельного предварительного изучения?

Методические принципы и приёмы

Дидактический материал по программированию можно разделить на две категории: 1) задачи на написание, отладку и тестирование программ, 2) упражнения и тесты. В упражнениях требуется записать элемент языка (например, оператор) или часть программы и (или) проанализировать их, то есть ответить на ряд вопросов, например, есть ли ошибки, объяснить их, всегда ли ошибка будет проявляться, как повлияет на результат какое-нибудь изменение в программе и т.п. Первый класс задач

является более важным, и его необходимо чаще практиковать. особенно при выполнении индивидуальных заданий. Но на начальном этапе изучения темы и при объяснении наиболее сложных вопросов нельзя игнорировать упражнения, которые играют подготовительную, вспомогательную роль, а также удобны при контроле и оценке знаний. Задания на программирование и упражнения необходимо разделять на два или три уровня сложности, что позволяет учесть способности обучаемых и объективнее оценить знания и умения.

Полезно использовать задачи и упражнения на составление и анализ **блок-схем,** которые почти не зависят от языка программирования. Используя свой опыт, авторы показывают, как и почему блок-схемы помогают писать программы. Их можно изучать как параллельно с языком программирования, так и последовательно. В первом случае составляется блок-схема алгоритма, а потом с её помощью разрабатывается программа. При последовательном изучении блок-схем и языка программирования сначала в течении нескольких подряд идущих занятий разрабатываются и анализируются только блок-схемы различных типов алгоритмов, а потом эти же типы алгоритмов изучаются на уровне программ.

На первых занятиях при изучении алгоритмов эффективным является составление и анализ блок-схем таких, например, «бытовых» алгоритмов, как алгоритм оплаты коммунальных услуг. Кроме того, на разных этапах изучения программирования можно рекомендовать следующие типы задач и упражнений: по готовой блок-схеме составить программу (часть программы, записать оператор и т. п.) или наоборот; по заданной схеме циклов написать одну или несколько из возможных постановок задач на определённую тему; с помощью блок-схемы объяснить, как работает фрагмент программы и другие.

Комплекс взаимосвязанных задач, однотипных в алгоритмическом отношении и одинаковых по трудности, позволяет показать, как некоторые детали условия могут существенно влиять на сам подход к решению. Примером таких являются, три матричные задачи для вычисления суммы чисел в каждой строке, в каждом столбце, во всей матрице.

Примером последовательности задач с усложнением являются следующие шесть задач: найти наибольшее число 1) из двух заданных чисел; 2) из трёх; 3) из произвольного количества; 4) найти наибольшее и наименьшее числа из n вариантов за один проход; 5) найти второе наибольшее число и количество его повторений: 6) найти наибольший элемент матрицы. В таком комплексе решение следующей задачи основывается на умении решать предыдущие.

Это эффективнее случайного набора задач, разделённых по сложности на несколько уровней. Если студент не может справиться с задачей

выбранной сложности, он может, не переключаясь на постановку принципиально другой задачи, «опуститься на один уровень ниже». И, наоборот, если задача окажется простой, он имеет возможность усложнить её. По каждой наиболее важной теме можно подготовить несколько таких последовательностей задач. И тогда этот принцип можно использовать при подготовке индивидуальных и домашних заданий, при проведении проверочных и контрольных работ, при подготовке к конкурсам, олимпиадам и другим соревнованиям. Тогда студент в состоянии сам оценить свой уровень. Оценка в таком случае должна зависеть от того, с задачами какого уровня он справился.

Комплекс взаимосвязанных задач можно использовать также при сравнении различных типов алгоритмов, а, значит, и различных операторов. Например, для демонстрации отличия циклов с известным и неизвестным количеством повторений можно предложить следующие три задачи. Найти сумму отрицательных чисел массива, если 1) массив не рассортирован; 2) массив рассортирован по возрастанию; 3) массив рассортирован по убыванию.

Полезно выполнять упражнения, в которых проверяется умение **«читать код».** С этого предлагается начать контроль и оценку знаний в первом семестре. В докладе приведено несколько вариантов фрагмента программы для работы с одномерным массивом, использующих оператор if в цикле. В вариантах, которые отличаются наличием или расстановкой фигурных скобок, а также, есть ли *else* для *if,* надо определить, что будет выведено.

Необходимо также обратить внимание на такие упражнения, в которых надо **сравнить** несколько вариантов программы или алгоритма. При этом необходимо выбрать правильный из предложенных и (или) объяснить ошибочные и записать верный вариант. Показано это на примере задачи «перевернуть одномерный массив».

Полезно использовать упражнения на выбор наилучшего оператора или фрагмента программы для решения одной и той же задачи. Например, поиск наибольшего и наименьшего элемента одномерного массива за один его просмотр эффективнее, чем за два.

В докладе рассматриваются и другие методические принципы и приемы: сравнение и повторение (операторы ветвления, операторы цикла, функции типа void и отличные от void, операции логическое умножение и сложение и т.п.), предварительная мотивация, многоуровневость, индивидуальные задания [1].

Заключение

Использование некоторых приемов, например, комплекса взаимосвязанных задач позволяет учесть уровень подготовленности обучаемых. Приведенные методические рекомендации полезны, прежде всего, при индивидуальной работе со студентами, у которых недостаточно развито алгоритмическое мышление. Таких, к сожалению, на нашем факультете немало. Прежде чем давать задание на отладку программ, желательно с помощью описанных приемов подготовить слабых студентов к разработке проектов.

Библиографические ссылки

1. *Аленский Н. А., Травин В. В.* Методика преподавания информатики: учеб.-метод. пособие с грифом УМО вузов РБ по естественно-научному образованию. Минск: «Адукацыя і выхаванне», 2019.