УДК 378.14

Каширина Надежда Валентиновна,

старший преподаватель,

ФГБОУ ВО «Государственный университет управления»;

Маран Михкель Михкелевич,

канд. техн. наук, доцент,

ФГБОУ ВО НИУ «Московский энергетический институт»,

г. Москва, Россия

НЕКОТОРЫЕ ВОПРОСЫ МЕТОДИКИ ПРЕПОДАВАНИЯ ПРОГРАММИРОВАНИЯ

В статье рассмотрены вопросы построения структуры дисциплины «Программирование» для студентов младших курсов. Подчеркивается важность тщательного анализа постановки задачи с выявлением особых случаев, возможных вариантов ответов и изучения базовых алгоритмов и способов их реализации на языке программирования с освоением лишь их базовых средств.

Ключевые слова: преподавание программирования, постановка задачи, алгоритмизация, проверка программ.

Nadezhda V. Kashirina,

Senior teacher,

FSBEI HE «State University of Management»;

Mihkel M. Maran,

PhD in technical sciences, assistant professor,

FSBEI HE NRU «Moscow Power Engineering Institute»,

Moscow, Russia

SOME QUESTIONS OF PROGRAMMING TEACHING METHODOLOGY

The article deals with the structure construction of the programming discipline for undergraduate students. In the article the author outlines the importance of a thorough analysis of the problem definition with the identification of special cases, possible answers and the study of basic algorithms and ways of their implementation in the programming language with capture of only their basic tools is emphasized.

Keywords: programming teaching, problem definition, algorithmization, program verification

Термин «Программирование для компьютера» можно понимать в узком и широком смысле:

- в узком смысле под этим понимается оформление известного алгоритма на заданном языке программирования для конкретной среды реализации. При этом алгоритм должен быть представлен известным программисту способом;
- в широком смысле под этим понимается решение прикладной задачи с помощью компьютера, начиная от содержательной постановки задачи до работающей в какой-то среде на каком-то языке программы.

В данной работе авторы предлагают понимать программирование в широком смысле, поэтому рассмотрим коротко все этапы, которые требуется пройти. Ограничимся рассмотрением проблем преподавания начального курса программирования, поэтому вопросы разработки программных комплексов для решения сложных и объемных задач останутся за пределами данной статьи. Эти вопросы подробно изложены в учебниках и монографиях по программной инженерии и технологии программирования и в [1].

Решение любой задачи начинается с вопросов «Что дано?» и «Что требуется найти?», могут быть заданы и дополнительные условия на исходные данные, метод решения. Процесс решения состоит из следующих этапов (ограничимся учебными задачами, поэтому ряд этапов жизненного цикла останутся за пределами нашего рассмотрения):

- анализ исходных данных и возможных результатов;
- разработка (выбор) метода решения;
- разработка алгоритма;
- составление программы;
- проверка составленной программы на тестах.

Анализ исходных данных и возможных результатов. Прежде, чем приступить к решению, студенты должны получить ответы на следующие вопросы.

1. При всех ли значениях исходных данных задача может быть решена или найдутся значения, при которых решение вообще невозможно (вопрос о корректности исходных данных, условия корректности)? Строго говоря, ограничения на значения исходных данных существуют всегда: в реальной жизни ни одна величина не изменяется от минимального допустимого в компьютере значения до максимально допустимого. Если решается задача, не имеющая физического содержания, то вопрос о корректности решить в общем случае невозможно. Но для студентов важно понимать, что исходные данные для задач с реальным содержанием всегда имеют ограничения на допустимые значения, и составляемые программы не должны допустить обработку заведомо неверных данных. Самый сложный случай, когда условия корректности имеются на сочетаниях данных – не может же 20-летний сотрудник иметь 25 лет трудового стажа! Конечно, студенты должны понимать и то, что не существует в принципе абсолютно надежных методов проверки исходных данных (если такие методы были, то их ввод бессмысленный!). Студенты должны с самого начала понимать, что любая программа должна сообщить о некорректности исходных данных и разъяснить пользователю, в чем она заключается.

2. При всех допустимых значениях решение пройдёт одинаково или ход решения зависит от значений. Простой пример: квадратное уравнение $ax^2 + bx + c = 0$. Какие ограничения накладываются на a, b, c для нахождения корней (без применения комплексных чисел)? Если дискриминант отрицательный, то исходные данные некорректны и уравнение вообще не может быть решено. А как быть, если $a < 0$ или $a = 0$? Задачу, очевидно, решить можно, но процесс решения должен быть усовершенствован.

3. Какие ответы возможны? Допустим, что дана последовательность чисел. Найти их сумму – проблем никаких, ответом будет одно значение. Найти

порядковый номер первого отрицательного числа – отрицательных чисел может не быть, что и должно быть сообщено пользователю. Это обстоятельство характерно для всех поисковых задач – следует предусмотреть отсутствие искомого. Соблюдайте принцип: ответ должен быть понятен пользователю БЕЗ дополнительных умозаключений. Найти сумму чисел, принадлежащих интервалу $[c_1, c_2]$. Таких чисел может не быть, что и должно быть сообщено пользователю. Число 0 – плохой ответ: во-первых, для его интерпретации требуются дополнительные рассуждения и, во-вторых, если $c_1 = -10$ и $c_2 = 10$, то искомая сумма может быть равна 0.

4. Решается ли задача однозначно или может быть несколько ответов. Допустим, что дана последовательность чисел. Найти значение максимального числа – проблем никаких, даже если все заданные числа равны между собой, то любое из них и есть максимальное. Найти номер(а) максимального(ых) числа(чисел). Максимальное число может встречаться несколько раз, в крайнем случае, все числа могут быть равны между собой, и все их номера являются ответом. Но если в постановке задачи указано найти номер первого (последнего, любого…) отрицательного числа, то вопрос о неоднозначности решения, очевидно, отпадает.

5. Решается ли задача однозначно или имеется неточность в постановке? Дана таблица чисел n строк и n столбцов. Если максимальное число находится выше главной диагонали, то …; если на главной диагонали, то …; если ниже, то … Если среди чисел максимальное значение встречается один раз или все они удовлетворяют приведённым требованиям, – то никаких проблем. Но если максимальное значение встречается два (или более) раз, но в разных частях таблицы, то задача не может быть решена однозначно. Простейший выход – дополнить постановку задачи фразой «максимальное значение единственное». Если исходные данные заданы так, что это условие не выполнено, то это уже вопрос корректности данных, о котором мы поговорили выше.

6. В каких пределах ожидается результат? В общем случае это определить сложно, но некоторые оценки часто возможны на уровне «здравого смысла»

(вряд ли средний рост человека равен 250 см или год рождения студента – 1955) или даже точнее (среднее арифметическое чисел, принадлежащих интервалу $[c_1, c_2]$, очевидно, принадлежит этому же интервалу). Студенты должны уметь оценить результат работы программы и помнить, что ничего не компрометирует программиста больше, чем программа, выдающая заведомо неверный результат.

7. Наш опыт свидетельствует, что нередко студенты неправильно выполняют переход от содержательных понятий «лучше – хуже» к математическим понятиям «больше – меньше». Например, при оценке спортивных результатов. Им следует понимать, что всегда в таких случаях требуется содержательный анализ, а не механически переход по принципу «лучше – значит больше».

8. Более сложными являются случаи, когда ограничения относятся к промежуточным результатам. Заданы объем резервуара, количество жидкости в нем, ожидаемые поступление и расход по часам в сутки. Определить, сколько жидкости будет в резервуаре после суток? Кажется все очень просто, но если на каком-то часе количество превышает объем резервуара, то поступление, очевидно, придется ограничить. И наоборот, если жидкость заканчивается, то расход придется ограничить. Правильная программа должна проанализировать такие случаи!

Разработка (выбор) метода решения. Различаем два случая: решение классической задачи из какой-то предметной области (математика, физика, …) и решение задачи, не относящейся к таковым. В первом случае метод решения известен, и остается лишь его реализовать. Если имеется несколько методов решения, то необходимо дополнительно провести анализ условий их применимости, преимуществ и недостатков и выбрать наилучший. Во втором случае метод придется разработать, тогда надо продумать, как можно эту задачу решить вручную, а затем заставить компьютер делать то же самое. Желательно, чтобы студенты предложили как можно больше способов решения и провели их сравнительный анализ. Важное свойство метода решения – его

вычислительная сложность: говоря немного упрощенно, зависимость требуемого количество операций от количества исходных данных в худшем случае. Желательно провести в первом приближении такой анализ и настроить студентов на выбор наилучшего по этому показателю метода решения.

Разработка алгоритма. Понятно, что алгоритмов существует огромное количество, им посвящена целая, можно сказать, энциклопедия, автором которой является Д. Кнут. Мы в начальном курсе программирования ставим более скромную задачу: ознакомить студентов с базовыми алгоритмами. К их числу относятся:

- Табулирование функции, с дополнительным условием или без.
- Нахождение сумм и произведений чисел заданной последовательности, с дополнительным условием иди без.
- Нахождение количества чисел последовательности, удовлетворяющих заданному условию, формирование из них новых последовательностей.
- Задачи поиска экстремальных значений в последовательности.
- Поиск элементов последовательности, удовлетворяющих условию (рассмотрением случаев, когда их нет, точно один или несколько).

Составление программы. Изучение этого раздела начинается с освоения минимального набора конструкций какого-либо языка программирования, достаточного для реализации перечисленных выше алгоритмов. Освоение тонкостей применения конструкций следует оставить на последующие стадии обучения. На каком именно языке пойти обучение, не так и важно. Долгие годы наилучшим для этого читался *Pascal*, специально созданный для обучения. По мнению авторов, этот язык хорошо подходит для этой цели, но с учетом того, что он на сегодняшний день не занимает лидирующих позиций в практическом программировании, можно рекомендовать язык *C#*, соединяющий в себе строгость языка *Pascal* и синтаксис *C++*. Это позволит студентам в дальнейшем легко перейти к самым распространенным на сегодня языкам *C++* или *Java*. Но строгость *C#* очень полезна на начальных стадиях обучения.

Упреки в том, что *C#* предназначен для объектно-ориентированного программирования (как и *Java*) считаем несущественными, потому что в нем имеются все средства и для традиционного процедурного программирования.

Любая программа состоит из следующих частей:

1. Объявление исходных данных.
2. Ввод исходных данных.
3. Решение задачи.
4. Вывод результатов.

Первая, вторая и четвертая части – стандартные, с минимальными вариациями, они образуют как бы каркас программы. Студенты должны хорошо освоить понятия типы и структуры данных, суть объявлений и операций ввода и вывода, их разницу. Освоение программирования начнем именно с этих средств.

Ограничимся данными: целое, вещественное, логическое. Важно, чтобы студенты освоили разницу между константами и переменными и между знаком равенства в математике и операцией присваивания в программировании. Далее рассмотрим ввод/вывод простых переменных, оператор цикла; и на этой основе можно решить задачи табулирования функций. Следует изучение одно- и двумерных массивов, их объявление, ввод/вывод. Для двумерного массива подчеркиваем возможность обработки (в том числе ввода/вывода) по строкам и по столбцам. После этого можно уже рассмотреть все перечисленные выше задачи. На языке *C#* имеются только динамические массивы, поэтому требуется их инициализация и написание циклов их обработки только через свойства (функции), позволяющие автоматически определить размерность массива. Это же обстоятельство должно быть студентами учтено и при формировании новых массивов: сначала определить количество элементов в них (может быть, и нуль), а лишь затем приступить к их формированию.

Важно обратить внимание на представление программы на экране, использование отступов для этого. Здесь большую помощь оказывают

современные среды (например, *Microsoft Visual Studio*), которые хорошо показывают это.

Выделение частей массивов по условию на его элементы одинаковое как для одно-, так и для двумерных массивов, и особого рассмотрения не требует. Для двумерных массивов требует особого рассмотрения реализация различных последовательностей обхода элементов и выделение частей массива по их расположению (выше – ниже главной (побочной) диагонали, ниже главной, но выше побочной диагонали и т.д.). Можно рекомендовать студентам написать таблицу индексов массива, например, 4 на 4, и самому найти закономерности изменения индексов для выделения той или иной части двумерного массива.

Важным вопросом является изучение средств структурирования программ – функций. Подчеркиваем два случая их использования:

- Для реализации частей программы, которые должны выполняться многократно с разными исходными данными.
- Как средство структурирования программы, при котором решения отдельных подзадач реализуются функциями и из них собирается программа решения всей задачи.

Для первого случая следует студентам подробно разъяснить способы передачи параметров и дать рекомендации по их применению, а также дать рекомендации по применению глобальных переменных.

Для второго использования функций рассмотрим метод функциональной декомпозиции как метод создания программ для решения сложных задач. Студенты должны понять, что декомпозиция означает выделение подзадач, каждая из которых имеет строго фиксированные исходные данные, результат и решаемую задачу. Решение выделенных подзадач всегда должно обеспечить решение любой исходной задачи. Если возможен случай, когда подзадача не может быть решена (например, из-за некорректности исходных данных для нее), то рекомендуем использовать сигнальную переменную. Тогда вызывающая программа может определить, получен ли ответ на подзадачу, и реагировать на это во избежание аварийной остановки.

Изучение объектно-ориентированного программирования возможно при достаточно большом объеме дисциплины. Важно понять принципиальную разницу между процедурным и объектно-ориентированным программированием. В первом случае исходим из решаемых задач, во втором случае – от обрабатываемых объектов. Класс – множество объектов, обладающих одинаковыми свойствами и одинаковым поведением. В простейшем случае можно создать класс – массив. Данные – это сам массив, могут быть какие-то его характеристики. Поведение – в простейшем случае операции ввода/вывода. Тогда в классах-наследниках могут быть решены самые разные задачи обработки.

При преподавании программирования возникает еще один важный вопрос: в какой степени использовать готовые средства решения задач. Имеем в виду стандартные алгоритмы *C++* для обработки последовательностей или язык *LINQ*, включенный в состав реализаций *C#* и *Basic*, позволяющих решить многие задачи, традиционно относимые к типовым алгоритмам. По мнению авторов, освоение базовых алгоритмов и их реализаций необходимо, но на заключительном этапе курса надо дать студентам представление и о современных средствах их решения, при применении которых целый отрывок программы может быть заменен вызовом одной функции.

Проверка составленной программы на тестах. Рекомендуем уже на стадии анализа задачи по перечисленным выше вопросам разработать и тесты, позволяющие проверить правильность работы создаваемой программы. Обязательно следует предусмотреть проверку программы на недопустимых данных. Любая программа должна либо выдать ответ, либо сообщить, по какой причине ответ не может быть получен. Понятно, что количество неправильных данных огромное и все не проверишь, но наиболее вероятные случаи предусмотреть надо. Корректность исходных данных, в общем случае, необходимое, но недостаточное условие успешного решения задачи. Поэтому в программах должны быть предусмотрены реакции на возникшие в ходе решения исключительные ситуации, и их тоже надо проверить тестами.

Программа, допускающая аварийный останов, – это брак программиста. Желательно еще до запуска программы проанализировать ее текст и получить ответы на вопросы (После перехода на диалоговый режим программирования анализу текста стали обращать незаслуженно мало внимания, а зря!). Эти вопросы подробнее рассмотрены в [2].

В статье были коротко рассмотрены проблемы построения начального курса программирования. Содержание дисциплины существенно зависит то количества выделенных для нее часов, а также от уровня подготовки студентов и целей обучения: какими именно средствами должен для освоения других дисциплин овладеть студент. В зависимости от этого, те или иные разделы могут быть пройдены более или менее глубоко. Авторы убеждены, что в любом случае студентов необходимо обучать именно решению задач от содержательной постановки через все этапы до проверки составленной программы. Какие при этом используются язык программирования и среда реализации, – это второстепенный вопрос.

СПИСОК ЛИТЕРАТУРЫ

1. Каширина Н.В., Маран М.М. Проблемы преподавания программной инженерии / Труды Международной научно-практической конференции «Информатизация инженерного образования» – ИНФОРИНО-2016 (12-13 апреля 2016, Москва). – Москва: Издательский дом МЭИ, 2016. – С. 146-150 [Электронный ресурс]. – Режим доступа: http://inforino2016.mpei.ru/doc/pr2016.pdf.

2. Маран М.М. Вопросы качества программного обеспечения в дисциплине «Программная инженерия» / ИНФОРИНО-2018. Материалы IV международной научно-практической конференции «Информатизация инженерного образования» (23-26 октября 2018, Москва). – Москва: Издательство МЭИ, 2018. – С. 146-149 [Электронный ресурс]. – Режим доступа: https://inforino.mpei.ru/submission/Documents/Inforino-2018_Russian.pdf.