УДК 004.421

В. А. Крайванова, **Е. Н. Крючкова**

Алтайский государственный технический университет
им. И. И. Ползунова
пр. Ленина, Барнаул, 656038, Россия

E-mail: krayvanova@yandex.ru; kruchkova_elena@mail.ru

ОЛИМПИАДНОЕ ПРОГРАММИРОВАНИЕ КАК ЭФФЕКТИВНЫЙ ИНСТРУМЕНТ ПОДГОТОВКИ ПРОФЕССИОНАЛЬНЫХ ПРОГРАММИСТОВ

Рассматривается применение принципов олимпиадного программирования в повседневной практике обучения студентов. Приводится перечень ключевых моментов фундаментального образования в процессе подготовки профессиональных программистов: производительность, размерность, типы данных, простота реализации, тестирование, командная разработка. Предлагаются примеры контрольных заданий с анализом часто встречающихся ошибок.

Ключевые слова: олимпиадное программирование, обучение, эффективный алгоритм, качество программных продуктов.

От мотивации к профессионализму

Современное развитие экономики выдвигает все более высокие требования к специалистам по разработке программного обеспечения. Однако в процессе подготовки программистов наблюдается существенный разрыв между теоретическими знаниями и применением их на практике. Зачастую большинство выпускников знают множество элементарных типов данных, но не видят разницы между 32- и 64-разрядными типами, знакомы со многими эффективными алгоритмами, но не могут определить, в каких ситуациях их следует применять, не имеют практических навыков оценки производительности созданного ими программного обеспечения (ПО). Вместе с тем эти факторы, наряду с другими аналогичными, определяют профессионализм программиста. Обширные теоретические знания в сочетании с отсутствием понимания и умения их применять на практике выливаются не только в отсутствие языковой грамотности и понимания собственного кода, но и в неочевидный и неестественный код, который невозможно сопровождать, проблемы ранней (и неэффективной) оптимизации, оверинжиниринг (решение простых задач сложными методами). Чтобы справиться с этими проблемами, выпускник вынужден фактически переучиваться.

Изучать различные аспекты и тонкости программирования необходимо даже в том случае, если они не используются в повседневной работе. Здесь возникают лишь трудности с освоением достаточно сложного материала, поэтому в обязательном порядке изучение таких аспектов должно даваться в университете, для фундаментального образования это необходимо.

Рассмотрим ключевые моменты, которым следует уделять особенное внимание в процессе обучения программированию:

- типы данных и особенности выполнения операций над ними;
- тестирование;

- размерность задачи;
- производительность;
- стандартные библиотеки.

Программисту необходимо понимать особенности простейших и векторных типов данных, их представление в памяти компьютера и специфику операций с ними. Программист должен: владеть навыками результативного тестирования и отладки программ, понимать необходимость этого этапа разработки ПО; уметь оценивать размерность задачи и выбирать достаточно эффективный алгоритм (оптимизация на стадии проектирования, а не кодирования); заботиться о технических характеристиках конечного продукта – времени работы алгоритмов и объеме занимаемой памяти. Эти характеристики должны быть достаточными для условия задачи. И, наконец, программист должен уметь применять стандартные библиотеки. Это, безусловно, экономит время самого программиста, а зачастую – и процессорное время.

Выделим правило KISS (Keep It Simple Software): необходимо делать программу настолько простой, насколько возможно в данной ситуации. Это принцип бритвы Оккама в применении к программированию. Чем проще программа, тем меньше в ней ошибок, тем элегантнее и понятнее код и тем легче его сопровождать.

Также немаловажным является формирование у студентов некоторых личностных качеств и навыков, необходимых программистам:

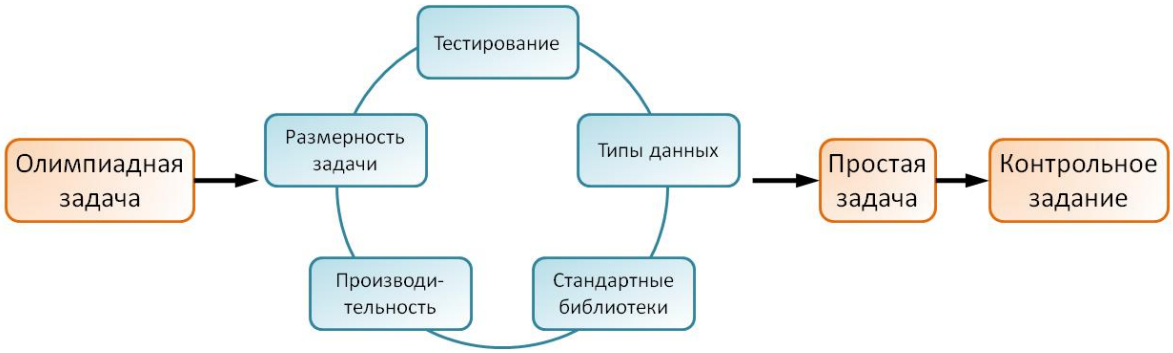
- стремление к максимальному контролю над кодом; уверенная формализация поставленных задач;
- нацеленность на успех, своевременное и эффективное решение поставленных задач;
- научно-техническое творчество и самостоятельность (в программировании любая сколько-нибудь крупная задача не является в чистом виде технической, из всех технических профессий программист – одна из наиболее творческих);
- навыки командной работы.

Успешно овладеть всеми описанными навыками и выработать у студентов необходимые профессиональные качества помогают олимпиады по программированию. Применение олимпиадных подходов в учебном процессе позволяет минимизировать описанные негативные эффекты у студентов и повысить уровень их готовности к профессиональной деятельности на момент выпуска. Олимпиады – это переход от обучения по стандартным учебным планам к расширению образовательной составляющей, увеличению количества и повышению качества усвоения основ и базовых навыков профессии. Это достигается за счет следующих особенностей олимпиадного движения.

1. Олимпиады вырабатывают четкий стиль программирования, умение тестировать программы, видеть возможные ошибки. Статистика показывает, что победители чемпионатов сдают задачи практически с первой попытки.
2. Командная работа на тренировках и на соревнованиях приводит к выработке приемов и навыков коллективного выполнения работ.
3. Одна из самых больших проблем подготовки программистов в России – отсутствие необходимого числа выпускников в области управления проектами. Нацеленность на успех – одна из несомненных ценностей олимпиад. Она одинакова для всех: и для тех, кто не победил, и для чемпионов. Студенты учатся работать так, чтобы в будущем суметь создать собственную команду и сделать ее успешной.
4. Олимпиады развивают способности к научному и техническому творчеству, самостоятельности, способности быстро ориентироваться в различных ситуациях.

Необходимо отметить, что многие олимпиадные методы обучения имеет смысл распространять на всех студентов, а не только на наиболее подготовленных. Безусловно, не следует требовать от основной массы студентов многочасовых ежедневных тренировок и навыков написания сложных алгоритмов без использования литературы, но следующие мероприятия полезны и необходимы всем без исключения программистам:

- требование к строгому соблюдению спецификаций ввода-вывода;
- строгое соблюдение ограничений на объем занимаемой памяти и время работы программы;



- решение простых задач за жестко ограниченное время (потребность в выполнении таких заданий вырабатывает у студента навык быстрого кодирования, операция кодирования не должна являться ключевой и самой сложной при решении задачи);
- автоматическое тестирование программ на тестах, содержание которых недоступно, а также последующая отладка (у студента вырабатывается понимание объективной необходимости тестирования программ, не обусловленной личностью преподавателя);
- решение задач и написание кода в команде (не менее 3 человек).

Навык качественного кодирования, лежащий в основе профессиональной деятельности программиста, нарабатывается исключительно практическим опытом. Научиться этому можно только на собственных ошибках. Олимпиадный подход преподавателя к разработке заданий имеет главную цель – дать студенту возможность учиться на собственных ошибках (см. рисунок). Начинать применение олимпиадных подходов к обучению программированию требуется с первого курса, так как именно в это время у студента формируется представление о процессе конструирования, отладки и тестирования программ. В данной работе мы рассмотрим некоторые примеры очень простых задач, решая которые первокурсники получают профессиональные навыки кодирования. Именно на простых задачах этот процесс проходит эффективно.

Примеры контрольных заданий

Задача A

Имя входного файла:	trim.in
Имя выходного файла:	trim.out
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Дан текст. Удалить из него все пробелы.

Формат входного файла. Входной файл содержит текст, состоящий из одной или более строк. Общая длина текста не превосходит 1 000 000 символов.

Формат выходного файла. В выходной файл необходимо записать текст с удаленными пробелами.

Пример trim.in	*Пример trim.out*
Сложная задача :) Десять раз подумай!	Сложнаязадача:) Десятьразподумай!

Рассмотрим характерные ошибки в задаче *A*.

1. Студенты часто описывают все переменные в одном месте, причем, как правило, в начале главной функции программы, так как классический подход требует избегать использования глобальных данных. Но выделение памяти под массив большого размера в теле функции приводит к переполнению стека и ошибке времени выполнения (*Runtime Error*).

2. Вторая ошибка, которую студенты часто допускают при написании подобных программ, – это выполнение сдвига текста «на месте» после каждого найденного пробела (что приводит к выполнению $O(n^2)$ операций, если текст состоит практически из одних пробелов).

3. Еще одна часто встречающаяся проблема – неэффективное использование библиотечной функции *strlen*(). Вычисление длины строки в цикле приводит к тому, что в результате программа работает $O(n^2)$ вместо кажущегося $O(n)$:

```
for (int i=0; i<strlen(s); i++)  {
        <обработка символа>
 }
```

Задача B

Имя входного файла:	palindrome.in
Имя выходного файла:	palindrome.out
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Дано десятичное целое число. Проверить, является ли оно палиндромом (перевертышем) в десятичной системе счисления.

Формат входного файла. Входной файл содержит единственное число, не превышающее 10^{10000}. Число может иметь лидирующие нули.

Формат выходного файла. В выходной файл необходимо записать слово YES или NO в зависимости от того, является число палиндромом или нет.

Пример palindrome.in	*Пример palindrome.out*
001221	YES
345	NO

Перечислим цели, достигаемые при решении задачи *B*:

- понимание того факта, что не любое число можно разместить в целом типе данных. В данной задаче число нужно хранить в виде строки;
- использование вещественных данных приведет к потере значащих разрядов;
- кроме того, требует проработки значительного числа частных случаев (лидирующие нули; четная и нечетная длина числа; число состоит из одной цифры; аккуратная работа с нулем – 0000 и 0).

Задача C

Имя входного файла:	sequence.in
Имя выходного файла:	sequence.out
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Дана последовательность целых неотрицательных чисел. Найти минимальное неотрицательное число, отсутствующее в последовательности.

Формат входного файла. В первой строке входного файла содержится N – количество заданных чисел, $N \le 10^6$. В одной или нескольких последующих строках содержится N чисел, каждое из которых не превышает 10^6. Числа могут повторяться.

Формат выходного файла. В выходной файл необходимо записать единственное число – ответ на задачу.

Пример sequence.in	*Пример sequence.out*
4 5 0 1 2 4	3

Цели, достигаемые при решении задачи *C*:

- понимание того факта, что миллион данных – это очень мало. Такой длины массив всегда можно создать. Большой массив не следует размещать в стеке;
- время решения $O(N^2)$ при $N = 1\ 000\ 000$ слишком велико;
- умение разрабатывать эффективные алгоритмы, используя традиционные методы;
- умение работать с массивами флагов.

Задача D

Имя входного файла:	set.in
Имя выходного файла:	set.out
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Даны два множества целых чисел. Найти пересечение этих множеств.

Формат входного файла. В первой строке входного файла содержатся $N_1 \le 10^6$ и $N_2 \le 10^6$ – количество чисел в первом и во втором множествах соответственно. Во второй строке содержится N_1 чисел первого множества, каждое из которых не превышает 10^9. В третьей строке содержится N_2 чисел второго множества, каждое из которых не превышает 10^9. Числа внутри множеств не могут повторяться.

Формат выходного файла. В выходной файл необходимо записать числа, входящие в пересечение описанных множеств.

Пример set.in	*Пример set.out*
6 8 4 50 1 2 3 8 10 8 1 16 3 5 100 4567	1 3 8

Кроме уже описанных ранее целей, задача *D* демонстрирует еще одну – разумное использование библиотечной функции *qsort*(). Использование этой функции позволяет решить задачу не за $O(n^2)$, а за $O(n\ logn)$, что является принципиально более эффективным. Написать код быстрой сортировки самостоятельно – задача сравнительно сложная. За время обучения студент, безусловно, должен реализовать стандартный алгоритм быстрой сортировки вручную. Однако на практике этого делать не стоит, так как все современные языки программирования содержат корректный и очень эффективный код для *qsort*(). Эта функция – несоизмеримо лучшая альтернатива пузырьковой сортировке, которая хоть и имеет принципиально более простой алгоритм, работает за $O(n^2)$ и зачастую сводит на нет весь выигрыш от работы с сортированными данными.

Задача E

Имя входного файла:	ariph.in
Имя выходного файла:	ariph.out
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Даны четыре целых числа: A, B, C и D. Вывести четыре числа: A + B, B – D, C * D и A / C (последнее число – с точностью до шести знаков).

Формат входного файла. Входной файл содержит четыре целых неотрицательных числа A, B, C и D, разделенных пробелами. Все числа по модулю не превосходят 1 500 000 000 (1,5 млрд), C не равно 0.

Формат выходного файла. В выходной файл необходимо записать одну строку, содержащую четыре числа A + B, B – D, C * D и A / C через пробел (последнее число – с точностью до шести знаков).

Пример aript.in	*Пример aript.out*
12 8 4 6	20 2 24 3.000000

Несмотря на кажущуюся простоту, задача *E* комплексно тестирует усвоение студентами специфики целочисленных и вещественных данных и особенности операций над ними.

Выводы

Принципы олимпиадного программирования необходимо применять в повседневной практике обучения всех без исключения студентов, которые получают профессию программиста. В этом случае преподаватели приобретают более разносторонний объем знаний и умений и, следовательно, могут обеспечить более высококачественную подготовку студентов.

Материал поступил в редколлегию 23.04.2012

V. A. Krayvanova, E. N. Kruchkova

OLYMPIAD PROGRAMMING AS AN EFFECTIVE TOOL FOR THE TRAINING OF PROFESSIONAL PROGRAMMERS

The principles of the Olympiad programming in the daily practice of training students are discussed in the article. The list of the most important and fundamental elements of education of professional developers is provided. This list contains such moments as the efficiency software, high dimensional data, data types, ease of implementation, testing, software development team. Some examples of problems and analysis of common errors are considered.

Keywords: olympiad programming, learning, efficient algorithm, the quality of software.