## Язык программирования

В базовой школе (8-9 класс) изучается школьный алгоритмический язык Кумир или Интал. Другие языки программирования могут изучаться на факультативах или в школах с нестандартными ПЭВМ. Цели изучения языка программирования можно сформулировать следующим образом: первая – показать школьникам, как конструкции алгоритмического языка могут быть выражены средствами языка программирования (иллюстративная цель); вторая – дать учащимся возможность проработать на ЭВМ алгоритмы, которые они освоили на уроках , особенностями программирования (прикладная цель). Этим определяются ответы на два важнейших вопроса: какой язык изучать и как построить его изучение? Прикладная цель диктует : изучать надо язык, имеющийся на доступной вычислительной технике; желательно, чтобы по форме и идеям он был как можно ближе к уже изученному алгоритмическому. Иллюстративная цель определяет методику изучения: язык программирования рассматривается как одна из форм записи алгоритмов, изложение ведется в постоянном сопоставлении с алгоритмическим языком. Необходимо выделить известные по алгоритмическому языку элементы, показать их общее в их сущности и различия в оформлении, выяснить особенности, ограничения и дополнительные возможности языка программирования. Рекомендуется придерживаться следующего плана:

1) основные элементы языка, постоянные и переменные величины, простые типы данных, выражения;
2) общая структура программы, описания, присваивание, ввод и вывод;
3) ветвления и циклы;
4) табличные величины;
5) вспомогательные алгоритмы;
6) дополнительные возможности языка программирования.

По этой схеме можно вести занятия в классе, на факультативе в кружке, разница лишь в степени подробности изучения. Учащиеся должны иметь общее представление о языке программирования, его назначении, сферах

применения, уметь правильно записывать на нем простейшие алгоритмы, запускать их на выполнение на конкретной ЭВМ. **При проведении занятий учитель должен иметь в виду, что подавляющему большинству учеников никогда не придется программировать. В основном они будут пользоваться готовым программным обеспечением.** Язык программирования в учебных пособиях излагается в качестве примера записи алгоритмов для испаолнителя ЭВМ.

Методически более правильным будет предложить учащимся уже написанные, готовые программы, ранее разбиравшихся алгоритмов. Эти программы можно “прокручивать” с различными данными, модифицировать, следя за работой конструкций языка. *Длина программы должна быть небольшой 10-15 команд.* ***Для экономии времени и охраны здоровья учеников желательно передавать заранее заготовленные программы по локальной сети с учительской на ученические машины.*** Большим подспорьем на уроке является демонстрационный телевизор. Все изменения в программе оперативно отражаются на демонстрационном экране, возможно коллективное обсуждение результатов.

## Решение задач

Способность эффективно применять ВТ при решении прикладных задач станет важным показателем квалификации специалиста. Сама постановка задачи – прерогатива человека, владеющего вычислительной техникой. Обращаясь за помощью к компьютеру, специалист должен знать особенности постановки решаемых им задач, представлять возможности машины, уметь правильно оценить результаты расчетов.

Как решать задачу? Этот вопрос не имеет однозначного ответа. Что здесь является наиболее важным: знание законов, теорем, формул; логическое мышление; интуиция; работоспособность? Не существует универсального алгоритма для решения задач. Однако были попытки дать общие рекомендации. Вопрос всегда оставался дискуссионным и вызывал споры. Декарт: «Расчлените любую изучаемую вами задачу на столько частей, на сколько

вы сможете и насколько это потребуется вам, чтобы их было легче решать». Лейбниц: «Это правило Декарта малоэффективно, поскольку искусство разделения … остается не поддающимся истолкованию … Разделяя задачу на неподходящие части, неопытный решающий может увеличить свои затруднения». Задачи должны не только и не столько способствовать закреплению знаний, тренировке в применении изучаемых законов, сколько формировать исследовательский стиль умственной деятельности. Нужно сформировать потребность максимального использования предыдущего опыта. Здесь имеется в виду то, что анализ задачи должен содержать вопрос: «Что есть общего у данной задачи с решаемыми ранее? Какие известные нам методы могут быть применены?» Этот вопрос один из самых полезных при анализе задачи. Если ученик находит нестандартное, оригинальное решение, то его следует приветствовать только в том случае, если такое решение требует меньшего объема вычислений или меньших затрат времени.

При обучении решению задач важное место занимает разбор готовых решений или, что практически эквивалентно по эффективности решение задачи от начала до конца учителем. Но опыт решения можно приобрести только при самостоятельной работе. Но даже на этом этапе одна из наиболее важных особенностей учителя – помогать ученику. С одной стороны нужно ученику предоставить максимальную свободу в работе, но нельзя оставлять наедине с задачей безо всякой помощи. Помощь учителя должна быть осторожной, ненавязчивой, но достаточной для продвижения вперед по ходу решения задачи. Такой подход, конечно требует от учителя знания индивидуальных возможностей и способностей каждого ученика, умения быстро ориентироваться, какая помощь необходима на данном этапе, что следует оставить для самостоятельной работы, доброжелательного отношения к своим ученикам – чтобы они не боялись лишний раз воспользоваться помощью.

## Принципы проверки учебных и олимпиадных задач по информатике

При обучении программированию в курсе информатики наиболее сложным для учащихся оказывается самопроверка (отладка и тестирование собственных программ). Дальнейшая же проверка правильности выполнения учебных заданий преподавателем, в свою очередь, является наиболее трудоемкой частью процесса обучения и предполагает использование методики будущего оценивания ожидаемых решений. Завершением этапа разработки программ является тестирование программы, т. е. выявление особенностей представленного решения на наборе специально подобранных тестов. Признавая очевидные преимущества проведения тестирования программ с целью оценки их работоспособности и эффективности, многие преподаватели искренне полагают, что результаты тестирования никогда не могут дать достоверной информации о качестве программы и уж тем более оценивать степень обученности учащихся. Тем не менее тестирование уже много лет используется в мире как средство оценки решений различных задач, и это ведет к совершенствованию систем тестов, появлению в данной области деятельности новых технологий. Но никакая система тестирования не может быть самодостаточным инструментом для проверки решений задач в процессе обучения программированию. Так как, если программа уже успешно прошла все тестовые испытания, преподавателю имеет смысл просмотреть ее листинг в присутствии ученика, отметить недостатки (если таковые имеются) в эффективности программной реализации тех или иных элементов алгоритма и в стиле программирования.

Если тестирование программы проводится интуитивно и без четкого плана испытаний, то этот процесс можно назвать искусством. Если же тестированию предшествует тщательный подбор данных для контрольных примеров, а само тестирование выполняется последовательно и аккуратно, то тестирование становится наукой. Тестовые испытания учебной программы направлены на то, чтобы проверить правильность работы программы во всех

предполагаемых практических ситуациях, оговоренных в условии задачи, а также оценить ее эффективность. при этом программа учащегося, с точки зрения преподавателя, является «черным ящиком». Проверку такой программы без непосредственного участия ученика можно провести лишь в том случае, когда задача поставлена четко и полностью. Особое внимание при этом следует уделять описанию форматов входных и выходных данных, типам входных величин и диапазонам их допустимых значений. Полезным является включение тестовых примеров входных и выходных данных в условие задачи. Эта мера позволяет ученику проверить, правильно ли он понимает задачу. В последнее время неотъемлимой частью формулировки условия учебной или олимпиадной задачи по программированию является явное указание на максимально допустимое время ее работы на любых данных, не противоречащих условию. Данное указание как раз и позволяет в дальнейшем оценить эффективность алгоритма, используемого при написании программы. Приведем пример неудачной и удачной формулировки условия учебной задачи с точки зрения ее дальнейшего тестирования.

*Условие 1.* Написать программу сортировки числового массива по неубыванию.

*Условие 2.* Написать программу сортировки одномерного массива, состоящего из N (N<=30000) целых чисел, каждое из которых по модулю не превосходит 32000, по неубыванию.

Входные данные вводятся с клавиатуры. В первой строке ввода находится число N – размер массива, в следующих строках N элементов массива, разделенных пробелами и/или символами перевода строки.

Время работы программы не одном тесте не должно превышать 3 с.

Пример входных данных:

3

4 –1

2

Возможный файл OUTPUT.TXT для приведенного примера выходных данных:

-1 2 4

Таким образом, постановка задачи (спецификация задания) должна обладать свойствами четкости и полноты.

## СИСТЕМА ТЕСТОВ

Рассмотрим описание тестовых данных, необходимых для оценки работоспособности и качества программ учащихся.

1. Первый тест должен быть максимально прост, так как его цель – проверить, работает ли программа вообще. Вполне допустимо в качестве первого теста использовать тест из условия задачи. Именно с помощью такого теста преподаватель сможет определить – верно ли понято учащимися условие задачи, соответствует ли программа описанным в условии форматам входных и выходных данных и правильно ли в ней названы входной и выходной файлы (если в условии задачи предполагается, что ввод и/или вывод осуществляется через файл). Если это не так, то при автоматической проверке учащемуся будет указан следующий вид ошибки: нарушение формата представления входных данных (Presentation Error).

2. Вторая группа тестов должна отслеживать так называемые ***вырожденные случаи***. То есть случаи, когда решение задачи или не существует (тогда в условии задачи должно быть оговорено, что именно должна сообщать программа в такой ситуации), или коренным образом отличается от основного алгоритма решения. В первую очередь к данной группе относятся *нулевые примеры*. Для числовых данных это обычно нулевые значения (конечно, если по условию задачи ноль является допустимым значением для той или иной переменной), а для текстовых – это пустой входной файл, а также последовательность из пробелов и/или символов перевода строки. Другой пример вырожденности – нарушение общности входных данных , требующее специальной их обработки. Так, если в учебной задаче требуется решить уравнение $ax^2 + bx + c = 0$ для любых действительных значений a,b,c, то при a=0 квадратное уравнение в общем случае становится линейным, что программа учащегося несомненно должна учитывать. Причем только в этом случае существенным становится равенство

нулю значения параметра b. Таким образом , для проверки данной задачи необходимы следующие тесты на «вырожденность»:

$$a = 0, \quad b = 0, \quad c = 0;$$
$$a = 0, \quad b = 0, \quad c = 1;$$
$$a = 0, \quad b = 1, \quad c = 2.$$

Если же в задаче входными параметрами являются координаты N точек на плоскости, а требуется, например, найти такую точку на той же плосткости, расстояние от которой до наиболее удаленной из N точек минимально, то, очевидно, , что при решении задачи случаи N=1 и N=2 следует рассматривать отдельно.

3. Следующая группа тестов должна проверять ***граничные случаи*** (входные и выходные данные принимают граничные значения). Основное назначение подобных тестов – обнаружить возможные программистские ошибки, которые могли быть сделаны учащимися при реализации в том числе и правильного алгоритма. прежде всего в данном случае следует проверить, что при написании программы для переменных были выбраны подходящие типы данных. Так, как если программа вычисляет суммы целых чисел, каждое из которых по модулю не превосходит 32 767, то есть для представления таких чисел можно использовать двухбайтовый знаковый тип данных (integer), то для вычисления результата такого типа уже явно недостаточно. Причем может даже оказаться, что получение точного результата возможно с помощью так называемой «длинной арифметики» – организации выполнения арифметических действий с помощью массивов, что существенно меняет сложность решения.

В задачах при решении которых используются численные методы, к граничным можно отнести тестовые данные, для обработки которых требуется максимальная точность вычислений, в частности, большие или очень маленькие, но отличные от нуля значения для входных параметров.

Часто критичным для программы является количество оперативной памяти, которое используется во время ее выполнения. Если все переменные располагаются в статической памяти, то наиболее распространенная ошибка

при программировании – выход за границу массива данных во время выполнения программы, причем появиться эта ошибка может зачастую лишь на входных данных максимального объема. Если же максимально допустимое количество данных по условию задачи таково, что программист должен использовать динамическую память, то обязательно должны присутствовать тесты, проверяющие программу на корректность работы с ними. Так, начинающий программист может неправильно выделять необходимую оперативную память во время работы программы и/или неаккуратно к ней обращаться. Также память в программе может использоваться неэффективно. В этом случае возможно подобрать тест, на котором программа учащегося работать не будет как раз в силу недостатка динамической памяти, хотя эталонная программа в подобных условиях функционирует нормально.

При работе со строками программа учащегося часто не может обрабатывать корректные с точки зрения условия задачи строки текста, длина которых превышает 255 символов. Это связано с особенностями используемого языка программирования или применением массива символов фиксированной длины вместо одной, двух переменных.

Если по условию задачи требуется обработать файл, состоящий из заранее неизвестного количества чисел или строк, то для многих программ критичным оказывается наличие пробелов после последнего числа и пустых строк в конце файла. Иногда при тестировании подобная небрежность в программировании намеренно прощается, тогда при подготовке текстов приходится следить, чтобы признак конца файла располагался непосредственно за последним значащим символом входных данных, причем обязательно в той же самой строке.

4. В следующую группу можно объединить тесты, проверяющие правильность алгоритма решения задачи в целом. Они делятся на общие тесты и тесты специального вида. Последние должны проверить работоспособность программы в случае специальной организации входных данных. Например, входные данные могут быть отсортированы сначала в порядке возрастания, а затем в

порядке убывания, хотя по условию определенный порядок над ними не предполагается, однако алгоритм решения может использовать сортировку входных величин. Или значения всех параметров можно сделать равными между собой. Помимо выявления специфики работы программ на таких тестах, они хороши тем, что зачастую позволяют проверяющему определить правильность выходных данных «вручную». Это уменьшает объем работы при проверке компьютерных результатов.

Общие тесты должны проверить все ветви логической схемы алгоритма, такая проверка называется ***испытанием ветвей.*** Однако если алгоритм решения задачи никак не зависит от особенностей входных данных, то есть логическое ветвление как таковое в алгоритме отсутствует, то в таком случае можно ограничиться одним-двумя общими тестами. Если же в программе или ее части подразумеваются два и более исходов, то каждый из возможных путей работы программы желательно проверить. К этой группе тестов относятся невырожденные примеры входных данных, для которых программа должна определять, что решение в задаче отсутствует, если подобная возможность оговорена в условии.

5. Если решение задачи может привести к ошибочному применению эвристических алгоритмов, то необходимо подобрать такие наборы тестовых данных, на которых такие приближенные решения будут работать неверно. Иногда для этого приходится рассматривать или даже реализовывать несколько различных эвристических подходов к решению задачи, подбирая для опровержения каждого из них свой собственный тест. Гораздо сложнее подобрать тестовый пример для случая, когда задача решается с помощью «перебора с предпочтением» и самостоятельно заканчивает свою работу, если время работы приближается к максимально допустимому. В этом случае говорят, что программа написана с отсечением во времени. Если перебор организован грамотно, то подобная программа зачастую быстро находит нужный вариант, а в дальнейшем незаконченным перебором лишь пытается доказать, что найденный вариант действительно лучший. К счастью,

подобные проблемы возникают в основном при тестировании олимпиадных задач, для которых описанный прием решения считается приемлимым.

6. Если прогон программы на предыдущих группах тестов показал ее правильность в целом, то тогда следует приступить к проверке эффективности используемых алгоритмов. Данное качество является наиболее уязвимым как в учебных программах, так и в решении олимпиадных задач по информатике. Упрощая для себя решение задачи или ее реализацию, учащиеся неоправданно увеличивают вычислительную сложность алгоритма, зачастую делая его непригодным для работы над большими, однако допустимыми по условию входными данными. Осознать данный факт самостоятельно они при этом не всегда могут. Соответственно возрастает значимость подобного рода тестирования.

7. Последнюю группу составляют так называемые случайные тесты. Такие тесты должны отражать реальные в полном объеме входные данные. Генерация входных данных для такой группы тестов (часто достаточно одного-двух тестов) не представляет труда. Однако проверка правильности работы программы учащегося сложна именно на таких тестах. Объясняется это тем, что в данном случае трудно проверить «вручную» результаты работы программы. Зачастую для этого необходимо написать специальную программу, по сложности не уступающую решению исходной задачи. Особенно это характерно для «большого случайного теста», которым завершается тестирование программы.