«Как стать чемпионом мира по программированию, или разбор полетов»

Авторы: *Е.А.Штыков, С.Я.Герништейн, М.Ф.Бакиров, С.Н.Васильев, А.В.Клепинин, С.В.Коган, С.В.Скорб, Н.Н.Шамгунов*

Аннотация: *В статье рассказывается о процессе подготовки и участия команды в олимпиаде по программированию. Приводятся различные принципы формирования команд и назначения ролей для каждого ее члена, а также обсуждаются возможные стратегии поведения команды на соревновании. Значительный акцент делается на обсуждении специфики самих олимпиад по командному программированию, принципиально отличающихся от обычной олимпиады, в которой базовой единицей является отдельно взятый человек, а не команда вцелом.*

Содержание:

Предисловие

Эта книга написана для будущих чемпионов мира по программированию. На ее написание нас подвигли собственные неудачи в данном вопросе и мы решили поступить по принципу: кто может - делает, кто не может - учит. Авторы были первыми, кто участвовал в чемпионате мира по программированию от Уральского госуниверситета. К сожалению, мы не можем сказать о себе "мы были первыми", а всего лишь "мы были восьмыми", но мы были там, мы "нюхали порох", и очень хочется, чтобы накопленный опыт не пропал, а обогащался из года в год. Идея книги была подсказана оргкомитетом регионального этапа чемпионата мира, который проводится ACM - Associating for computing machinery - с 1977 года. Только в 1996 это проведение докатилось до Урала. Члены оргкомитета, состоявшего из москвичей и ленинградцев выпустили некую аналогичную брошюру. Внимательно ее проанализировав, авторы определили ее непригодность для большинства команд страны в качестве руководства по подготовке. Не подлежит сомнению, что и наш опус будет полезен не каждому читателю. Чтобы читателю было легче разобраться, для него или не для нет эта книга, мы стараемся как можно чаще приводить обоснования наших утверждений, или хотя бы примеры из жизни, заставившие нас думать так, как мы пишем.

Мы надеемся, что материал, изложенный в этой книге мог бы стать подспорьем многим ВУЗам Урала и Сибири при подготовке своих команд. Серьезное отношение к такой подготовке не только принесет пользу студентам-участникам в плане изучения компьютерных наук, программирования, дисциплинированности и умения работать в команде (качество, ценность которого стали осознавать даже современные российские менеджеры), но поможет сделать хоть какие-то шаги в устаревшей, на наш взгляд, позиции дистанцирования Москвы и Ленинграда от так называемой "периферии" - остальной части России. Одной из целей, которую мы ставили, было показать, что даже такое не слабое дело, как победа на чемпионате мира, может быть по плечу тем, кого столичные преподаватели не считают себе конкурентами.

Наиболее полезные идеи книги были подсказаны или отредактированы Дмитрием Олеговичем Филимоненковым, остальное мы придумали сами.

Также авторы выражают глубокую личную благодарность Алексею Руфимовичу Данилину за общие ценные указания и снабжение литературой, декану факультета Магазу Оразкимовичу Асанову и ректору университета Владимиру Евгеньевичу Третьякову за всемерное содействие в подготовке команды, Хасану Салмановичу Сумгаипову, Анатолию Васильевичу Макарову и Областному комитету по делам молодежи, поддержка которых и сделала вообще возможной нашу поездку.

Осознание себя как команды

1. Вступительные общие слова.

Тезис: три идеальных программиста, случайно собравшихся вместе в одно время, представляют собой более слабую команду, чем та, где, может и вовсе не быть "звезд", но которая специально тренировалась. Иллюстрацией этого утверждения может служить задача о поджаривании бифштексов. Пусть задана сковорода, на которой может быть размещено от нуля до четырех бифштексов. Каждый бифштекс в произвольный момент времени может быть перемещен на сковороду, удален с нее, или перевернут (бифштекс имеет две стороны). Требуется поджарить шесть бифштексов за минимальное время; бифштекс считается поджаренным, если он обжаривался в течение 15 минут с каждой стороны. Решение: жарим четыре бифштекса с одной стороны, затем два переворачиваем, а два полуподжаренных заменяем на два сырых, затем два уже готовых убираем, два оставшихся переворачиваем и докладываем два, отложенных прежде. Итого уходит 45 минут. Этот результат недостижим, если бифштексы не будут работать как единая команда, а каждый будет стараться поджариться независимо от других. Заметим, что этой "команде" приходилось для получения оптимального результата проделывать не совсем "естественные" действия: уходить со сковороды недожаренным. Как ни проста эта задача, она демонстрирует важность наличия между членами команды некоторых дополнительных мотивов действия, отличающих команду от простого набора. Несколько утрируя идею, ее можно выразить такой поговоркой:

У игрока - члена команды - постоянно возникает выбор: браться за ту или иную работу самому, поручить другому, или эту работу вообще не стоит выполнять. Однозначной рекомендации для принятия такого выбора мы не знаем. Один из способов облегчить себе жизнь - договориться заранее, кто какую работу выполняет и какая работа считается ненужной. Сформулировав такие общие критерии, команда обезопасит себя от проблемы выбора во время тура. При этом возникает другая проблема. Чтобы не рассуждать и при этом не ошибаться, формальные критерии выбора должны быть записаны чрезвычайно точно и одинаково поняты всеми игроками (участниками). Если круг задач, которые предполагается решать, широк, то такие критерии едва ли могут быть записаны, и вместо них используются упрощенные критерии, оставляющие, с одной стороны, определенную свободу выбора, и с другой, дающие ответ во многих ситуациях. Проводимые тренировки позволяют игрокам накопить опыт в принятии подобных решений, в результате чего команда начинает представлять из себя единое целое. Набор правил, помогающих принимать решения по поводу своих и чужих действий, будем далее называть стратегией. Стратегия подобна таблице умножения - не решая проблемы в целом, она подсказывает ответ в простых ситуациях.

Далее рассмотрим известные нам стратегии, выделяя у каждой плюсы, минусы, и отмечая, для каких команд более всего предназначена та или иная стратегия.

2. "Три мудреца в одном тазу ...".

Сразу после старта каждый участник выбирает себе задачу, с которой в состоянии справиться, и начинает решать ее на отдельно взятом листочке. Выбравший себе самую простую задачу может решать сразу на компьютере. Для своей задачи каждый

сам придумывает тесты, осуществляет отладку и сдачу. После посылки задачи на проверку жюри, участник за компьютером меняется, освободившийся выбирает себе новую задачу, и так далее. Достоинством стратегии является то, что все простые задачи будут решены (здесь и далее, слова "будут решены" означают, что сама стратегия направлена на решение этих задач, что никакие другие не могут быть решены, пока с простыми не покончено). Как показывает опыт, иногда этого достаточно для победы. Далее, когда двое будут решать две последних простых задачи, третий игрок уже освободится, и сможет прорабатывать более сложную; освободившись, к нему может подключиться второй, а при необходимости, и третий, так что оставшееся после решения простых задач время команда может варьировать, чтобы в соответствии со своими возможностями решить еще одну или две задачи.

Нетрудно показать, что простые задачи выгоднее решать раньше. В самом деле, если сложная задача так и не будет решена, то предпочтительнее иметь результат, меньший по времени. Решенные на ранней стадии тура простые задачи дают меньшее время, чем те же задачи, решенные позднее - их вклад будет содержать в себе время на решение более сложных задач, по предположению о сложности - большее время. Данная стратегия всецело направляет игроков на решение задач в правильном порядке - от простых к сложным. Наряду с этим, она обладает рядом недостатков.

Во-первых, все три участника должны виртуозно владеть всеми стадиями программирования - от написания формальной постановки до завершающего тестирования; затягивая решение своей задачи, игрок задерживает еще сразу две другие задачи, т.е. каждое время простоя (придумывание тестов, дополнительная отладка после неудачных попыток сдать задачу) умножается как минимум на три. Наряду с этим, как минимум один из игроков должен обладать глубокой математической подготовкой, что редко совместимо в одном человеке. Наконец, так как нет конкретного человека, принимающего решения о распределении работ, команда может (и, часто так и будет) оказываться в цейтноте: идея о консолидации сил для "добивания" одной, последней, задачи появится слишком поздно; пропадут впустую усилия двоих, решавших каждый свою задачу.

Кроме того, стратегия предполагает существенные накладные расходы при любых попытках взаимопомощи: человек, занятый (или который только что был занят) решением собственной задачи, часто не в состоянии перебороть инертность мозга, и переключиться на другую задачу.

Между игроками должно также иметь место очень хорошее понимание друг друга на компьютере: настройки среды программирования, клавиатуры, редактора, лучше иметь общие.

В целом, стратегия кажется разумной для трех "звезд" - игроков, каждый из которых владеет программированием во всех стадиях, и дополнительно имеет глубокую математическую подготовку. Виртуозное владение техникой программирования и его математическим аппаратом дадут команде большое преимущество и позволят успешно выступить, если команда обладает также высокой коммуникабельностью. Но команды, сочетающие в себе все перечисленные качества, встречаются редко.

3. "Столичная" система.

Такое название стратегия получила потому, что была впервые в России опубликована в информационной методичке регионального этапа чемпионата мира по программированию ACM членом жюри Е.Андреевой. Во главе команды находится "диктатор" - генератор идей; кроме него, команда включает "реализатора" и "тестера", причем последние две роли могут меняться во время тура. Диктатор излагает алгоритмы решения задач, которые поступают на доработку к одному из участников (возможно, и к самому диктатору). Реализатор, он же - главный программист, - занимается воплощением изложенных и доведенных до более или менее низкого уровня идей в машинный текст. К моменту окончания его работы над очередной задачей тестер уже должен иметь готовый набор тестов, позволяющий быстро проводить отладку и тестирование программы. После этого программа посылается жюри, а команда переходит к следующей задаче, идея решения которой уже некоторым образом оформлена диктатором.

К достоинствам стратегии относится то, что команда будет разумным образом управляться, так как есть участник, ответственный за это, и потому управлению будет уделено должное внимание. Более того, такая команда имеет все шансы решить все простые задачи за практически минимально достижимое время, и иногда уже этого будет хватать для победы. К тому же моменту, когда простые задачи будут решены, у вех будут свободные руки, а на бумаге - почти наверняка готов проект решения очередной, более сложной задачи. Если диктатор обладает глубокими познаниями в математике (а чаще всего, так и бывает), то команда имеет высокие шансы на решение математических задач турнира. Ролевое распределение во время тура решает задачу выбора работы для игроков. Наконец, такая команда едва ли может попасть в цейтнот в концовке, так как диктатор на чеку.

Недостатком схемы является чрезвычайно большая ответственность, которая ложится на диктатора. На нем - стыд или слава за все принятые решения, как по решению задач, так и по распределению работ. Человек, принимающий на себя обязанности диктатора, должен быть весьма волевым, обладать большим личным опытом участия в соревнованиях, олимпиадах. Но даже если такого человека удалось, отыскать (а это не так просто), каждая его ошибка будет сокрушительным ударом по собственной команде. А ведь помимо решения задач, в том числе, математических, на диктаторе лежит ответственность за оценку стоимость разработки тех или иных идей; когда задачи, посланные жюри, начнут возвращаться не зачтенными, ему придется экстренно решать, как с ними быть - продолжать отладку сейчас или отложить, прерывать решение текущей задачи или вставить возвращенную задачу "в очередь". Помимо перечисленного, такая тактика предполагает, в целом, последовательное решение задач. Последовательный подход, безусловно, оправдан, когда речь идет о простых задачах, - таких, где на разработку алгоритма уходит 10 - 15 минут. Эти задачи следует решать за минимальное время и затраты на организацию параллельных работ в любом виде едва ли окупятся. Однако, параллельное решение может оказаться выгодным для задач, требующих 1,5 - 2 человеко-часа. Данный же подход не предусматривает распараллеливания, кроме как на капитана и двух других членов.

4. "Вместе весело шагать..."

Для контраста хочется привести еще пример-антипод для стратегии, где каждый решал свою задачу. Можно, наоборот, решать все задачи вместе, начиная с простых и постепенно переходя к сложным. При этом возникнет естественное ролевое распределение игроков внутри одной задачи; игроки будут последовательно выполнят этапы реализации задачи, как они себе их представляют. Нет нужды говорить, что такая команда обязана процесс программирования представлять себе одинаково.

Недостатки стратегии явно видны: потери времени при решении задач никем не контролируются и могут принимать большие размеры. Параллельная работа над разными задачами отсутствует, что снижает общее число решенных задач; с другой стороны, неконтролируемые временные потери ухудшают время зачета сданных задач.

Тем не менее, стратегия имеет и ряд достоинств, о которых не следует забывать тем, кто разрабатывает "лучшие" стратегии. Во-первых, за время всего тура, скорее всего, будут решены все простые задачи, и при этом, команда достаточно застрахована (если взаимопонимание между партнерами на высоком уровне, а это широко распространенное качество) как от ошибок в определении простоты задачи, так и от ошибок отдельных игроков; отдельные ошибки имеют высокую вероятность исправления другими членами команды. Во-вторых, наработка взаимопонимания, достаточного для применения такой стратегии, требует весьма малого количества тренировок, лишь бы команды была "спаяна" достаточно крепко. В-третьих, для формирования команды вообще не требуются "звезды" - асы в программировании и математике.

Более того, такая стратегия ценна еще и тем, что может быть применена большинством команд, как на тренировке, так и на соревнованиях. Ее применение будет всегда оправдано, если известно, что решения всех простых задач достаточно для прохождения в следующий круг соревнований. Другой пример, - когда ВУЗ выступает впервые и время для тренировок весьма ограничено (или это может быть отборочный чемпионат ВУЗа). Если команда решит для себя, что показать более, чем 50-ти процентный результат, занять место в первой четверти турнирной таблицы для нее важнее, чем рискнуть на первое место и проиграть, простая стратегия может стать оптимальным оружием.

5. "Сумасшедшее чаепитие".

Эта стратегия, как и одна из приведенных выше, является ролевой. Остановимся на ней подробнее, так именно ее применяли команды Уральского университета в своей первой поездке, именно она была выстрадана этими командами и о ней мы можем сказать довольно много. (Название, безусловно, не лучшее, заимствовано из книги Кэрролла "Алиса в стране чудес"). Тот факт, что весьма разные команды, применявшие эту стратегию, показали весьма близкие результаты, позволяет предположить, что результат есть не только следствие выбора команды, но и следствие выбора стратегии (это утверждение - попытка обосновать, почему столько внимания уделяется этому вопросу). Команда должна быть дружной и иметь высокий уровень коммуникабельности. Ничто не приносит столько бед, как прохладный психологический климат. Студенты на чемпионате подобны пилотам в кабине пассажирского лайнера - если между пилотами плохая атмосфера, лететь нельзя.

Вместе с тем, каждый человек индивидуален, не похож на других, и ему приятно, если для реализации себя, в том числе, на соревнованиях, у него есть своя ниша. Тем самым высказана мысль, что люди, объединенные в одну команду, вполне могут обладать разными талантами. Для команды нам потребуется один ярко выраженный математик, один такой же программист, и один, в более или менее равной степени владеющий обоими ремеслами. Поясним чуть подробнее.

Математик призван видеть абстрактную суть вещей. Его стихия - обобщать, выписывать рекуррентные формулы, дифференцировать, доказывать оценки численных методов, получать функции общего члена, подсчитывать число сочетаний и порядок сходимости, брать первообразные и первые интегралы, работать алгебраическими методами над геометрическими фигурами и наоборот, выполнять дополнительные построения, применять разностные схемы. Его место там, где волшебная палочка математического аппарата превращает грозное условие в послушную двухпараметрическую игрушку. Он способен сделать бесконечное число шагов за ничтожное время, написав формулу суммы ряда или ортогонализацию Грамма-Шмидта. Произведения, суммы, цепные дроби, векторная алгебра, геометрические методы, - да разве можно перечислить весь его арсенал.

Программист являет собой в некотором смысле противоположность математику. Он мыслит конкретно, его сырье - графы, цепи, циклы, очереди, стеки, массивы, списки, деревья, деки, правила вывода, ветвления, потоки, файлы. Так же как и у математика, его вселенная необъятна и абстрактна, но каждая ее деталь подразумевает конкретное воплощение. Это огромная мастерская. Стоит программисту определить, какой инструмент ему нужен, и начинается сборка, и вот уже слышно веселое жужжание новорожденного механизма. Там, где математик видит нечто, ни на что не похожее, или где с его губ падает слово "тривиально", программист может найти по горло работы. Виртуоз отладки, гений трансляции - программист понимает каждое движение курсора, каждый машинный "вздох" и паузу не хуже, чем восклицания и плач партнеров. В душе, каждый программист немного хакер, но он старательно скрывает в себе это качество и может даже выучить наизусть теорию матроидов, если ему скажут, что это всего лишь объекты, двойственные графам.

Наконец, оставшийся участник команды, "мастер на все руки", или "связной" принимает на свои плечи всю тяжесть непосредственного решения задач. Несмотря на то, что могучий дуэт математика и программиста способен смести горы на своем пути, оба они могут потерять немало груза из-за крошеной дыры в рюкзаке, забыть завтрак внутри при сворачивании палатки, и т.п. Когда они разрабатывают железнодорожное полотно, для них не существуют отдельные рельсы, болты и гайки. Программист виртуозно выполнит стальную магистраль, но первые же пассажиры могут никуда не доехать, ибо на семафоре не будет предусмотрено переключение сигнала, а на станциях - открывание дверей.

Иначе говоря, творцы часто считают слишком многие вещи само собой разумеющимся, и железная болванка "Intel" не всегда способна простить им это. "Практик" кует железо, из которого потом рождаются строки кода под умелыми руками программиста; он спускает на землю написанные без разбиения на строки математические формулы; наконец, одна из самых важных его миссий, - он позволяет математику не отвлекаться на "ерунду" вроде составления тестов. И последнее: если программист в команде устанет, его мозг и пальцы перестанут действовать идеально

согласованно, и еще в некоторых критических ситуациях, практик способен сеть за штурвал-клавиатуру и спасти судно.

Помимо перечисленных личностей, в команде должен иметь место капитан, - игрок, пользующийся среди друзей наибольшим доверием. Его функции, дополнительно к прямому назначению, будут описаны чуть позже. Пока отметим лишь, что желательно, чтобы капитан не был программистом, так как эта стратегия не предполагает знакомства программиста со всеми задачами, а это либо снижает компетентность его решений, либо требует дополнительных затрат на ознакомление программиста и задач.

После старта ближайшая цель команды - разделаться с простыми задачами. Следует, однако, понимать, что при таком разделении труда, какое описывается, простые задачи никогда не будут решены за минимально возможное время и весьма часто найдется команда, сдавшая эти задачи раньше. Не нужно горевать об этом. Нужно побеждать общим числом задач, или даже временем, но на следующем уровне. Чтобы этого достичь, нужно минимизировать время простоя отдельных игроков. В соответствии с этой целью, программист немедленно начинает набивать общий шаблон: ввод, обработка, вывод. Он же несет обязанность немедленной отладки шаблона, так как ошибку в шаблоне придется исправлять многократно. В силу того, что применение описываемой тактики требует существенного числа тренировок, программист, как правило, будет знать искомый шаблон наизусть к началу соревнований. Еще раз подчеркнем, что кроме специально оговоренных ниже случаев никто не должен лезть за клавиатуру. Компьютер - дом программиста, и никого больше.

В то время, как программист занимается набором шаблона, математик и практик изучают условия задач. Цель этого изучения: обеспечить программиста и практика двумя самыми простыми задачами. Самая простая задача выдается программисту (обычно есть сомнения, и, когда простые задачи отделены, их список с минимальными комментариями представляется практиком программисту, и тот выбирает себе самую простую задачу). Следующая по простоте задача передается практику, который на более или менее низкоуровневом псевдоязыке излагает с ходу ее решение. Пока эти двое (практик и программист) занимаются описанным процессом выбора двух задач, математик выбирает следующую по простоте задачу и принимает решение: оставить ее двум другим или взять себе. Нет однозначного рецепта для любой ситуации. Главное, что должен обеспечить математик своим решением - минимизацию простоя. К тому моменту, как программист закончит свою задачу, практик уже составит к ней некоторые тесты. В то время, как программист далее будет, обратно, тестировать свою задачу, а потом разбираться с задачей практика, практик должен иметь возможность приступить к третьей задаче. Это может быть либо никем не тронутая простая задача, либо математическая задача, разобранная математиком по косточкам, либо более сложная задача. Последняя ситуация соответствует случаю, когда простых задач не осталось, и математик собирается поменяться с практиком местами, когда участь первых двух задач будет решена.

В общем и целом, математик должен обеспечить практику задание, не пересекающееся с собственной работой математика на момент окончания решения первой задачи. Поскольку на решение одной задачи не мобилизуются все силы команды, первая (самая простая) задача может быть решена даже в 1.5 раза позже,

чем при использовании предыдущих двух стратегий. Однако, надо отдавать себе отчет, что эта задача должна быть решена несколько быстрее, чем за час (эта фраза имеет ввиду, что к такому времени нужно стремиться на тренировках, но нет никакого смысла убиваться по этому часу на самих соревнованиях).

То обстоятельство, что за компьютером все время сидит один человек – программист – требует от двух остальных особой аккуратности в оформлении решений задач. Существует два возможных подхода: либо программист самостоятельно доводит данный ему алгоритм до программного кода; либо составитель алгоритма (математик или практик) помогает ему в этом. С точки зрения максимального распараллеливания работы первый подход эффективнее, но в действительности, второй вполне может приводить к лучшему результату, если только реализуемый алгоритм не является совершенно тривиальным.

По-видимому, наиболее разумным будет оставить за программистом решение, будет ли он самостоятельно реализовывать данный ему алгоритм или воспользуется помощью автора. Возможно, программист решит набрать часть алгоритма самостоятельно и воспользоваться помощью товарища при наборе другой части программы.

Итак, процесс пошел, осталось его поддерживать. Необходимость поддержания возникает не всегда, а когда решена очередная задача. Назовем такой момент точкой выбора, если до конца тура осталось более двух часов, и критической точкой, если менее 70 минут. В диапазоне 80 - 110 минут такая точка может быть как точкой выбора, так и критической - в зависимости от того, какие задачи остались, сколько решено, и какие находятся в разработке. Решение о том, является точка критической, или нет, должен принять капитан.

Если точка - точка выбора, то освободившийся оценивает текущую ситуацию, и, в зависимости от числа задач в разработке и степени их завершенности, начинает разрабатывать новую задачу, приготовленную для него математиком, придумывать тесты, если это не очень трудоемко, к задаче, которая вот-вот будет набита и отлажена; наконец, готовить работу практику, если это математик и перечисленные варианты почему-либо не устраивают.

Если точка является критической, и освободившийся один, то он либо присоединяется к решению задачи, находящейся в разработке, либо берется за прописывание в псевдокоде задачи с ясным алгоритмом - по решению капитана. В этот же момент может быть принято решение об использовании эвристики.

Отметим мимоходом, что наряду с отсеиванием простых задач для решения в первую очередь, жизненно необходимо отсеивать также сложные задачи, которые решать вообще не планируется. Новая задача ни при каких условиях не может быть выбрана из этого "черного списка", если хоть одна из других задач еще находится в разработке. Смысл этого условия - в минимизации времени решения. Выше головы не прыгнешь, а разбазаривание ресурсов на решение трудных задач, как уже отмечалось, увеличивает суммарное время зачета для задач, которые будут в конце концов сданы командой.

Если точка является критической и освободившихся двое, то капитан принимает решение - добивать оставшуюся в разработке задачу, или есть все шансы за успех решения еще одной.

Неудачной можно назвать ситуацию, когда очередная задача готова и сдана, но алгоритма для следующей еще не придумано. Такое может случиться как вследствие ошибок планирования или неправильного выбора задач, так и в результате "естественных" причин (например, одна простая задача, а остальные относительно сложные). В этом случае программисту приходится "заниматься не своим делом" – идти на помощь одному (или сразу обоим) из оставшихся членов команды. Решение о том, чем именно программисту заняться, принадлежит капитану команды.

Описанная выше ситуация плоха по многим причинам. Во-первых, программист настроен на набивание текста, отладку, работу за компьютером, и внезапное перемещение его на роль составителя алгоритмов не способствует улучшению производительности команды. Во-вторых, программиста придется "вводить в курс дела" – рассказать ему условие и варианты решений, над которыми ведется работа, а это опять-таки потеря драгоценного времени. И в-третьих, компьютер у команды один, и он является достаточно важным, если не важнейшим ресурсом, которым владеет команда, и поэтому простой компьютера крайне нежелателен.

В общем и целом, вся ориентация данной стратегии - на минимизацию простоя, на постоянное решение задач, имеющих реальные шансы быть завершенными до финального свистка.

Стратегия, как и остальные, имеет много плюсов и минусов. Наиболее важный плюс: разделение труда не только ролевое, но и функциональное, что позволяет минимизировать простой без яркого "лидера". Наиболее важный минус: нет шансов на победу, если не смогли решить хотя бы одну или две задачи из списка "более сложных". Такая стратегия на решении только простых задач проигрывает по времени всегда.

Начав тренировки по данной стратегии, любая команда довольно быстро найдет оптимизирующие изменения, которые позволят более эффективно работать именно этой команде. Поэтому нет смысла далее подробно прописывать "программу" действий отдельных игроков. То, что было хорошо для первых команд УрГУ, совсем не обязательно будет приемлемо для всех остальных.

Условное название стратегии выбрано из-за того, что "освободившийся" игрок поступает как типичный мартовский заяц: ищет относительно чистую чашку.

6. "Предохранитель".

Еще одна ролевая стратегия, являющаяся разновидностью предыдущей. В окружающем нас мире многие любят заниматься одним (и только одним) делом. Поэтому часто существенно сложнее отыскать хорошего "практика", чем математика или программиста. Предлагаемая к рассмотрению техника предлагает один из вариантов решения этой проблемы. Количество приведенных стратегий не должно раздражать читателя. Едва ли возможно поверить, что для столь молодого в нашей стране дела может сразу быть выписан оптимальный для всех алгоритм. Поэтому

следует выбрать моменты различных стратегий, кажущиеся наиболее разумными, и объединить их в свою собственную, "победную".

Рассматриваемая стратегия выделяет в команде одного математика и двух программистов. Кроме того, один из членов команды является капитаном. Сразу после старта каждый из программистов получает по простой задаче (Методы выбора простых задач могут быть различны; например, каждый может читать с разных концов до простой, или один садится набивать шаблон, а двое читают, и т.п. Во всяком случае, эта тема достаточно подробно обсуждалась выше и ничего нового в этот момент стратегия не вносит). Далее одна задача набирается, а другая параллельно готовится. При этом математик служит как бы "центром", хозяйственной частью команды. Он занимается составлением тестов, помогает каждому из программистов в изобретении алгоритма, при необходимости, проводит оценки времени, прописывает решения, алгоритмы для задач на бумаге. Программисты регулярно обращаются к нему за помощью, тестами и консультациями.

Отличительная особенность поведения такой команды - смена оператора во время тура. Каждый программист ведет "свои" задачи. Такой подход имеет свои плюсы и свои минусы. Плюсы: облегчение загрузки программиста. Не каждый способен выдержать пять-шесть часов напряженной работы за терминалом без потери производительности, внимательности, сообразительности. А такая потеря дорого обходится: это и снижение эффективности использования компьютера, и проблемы в конце тура, когда бывает чрезвычайно важно сдать еще одну задачу. Здесь же каждый программист "предохраняет" другого, позволяет ему немного отдохнуть. Другое преимущество: локализуется ответственность за сдачу конкретной задачи. Т.е., математик в курсе событий, он способен квалифицированно охарактеризовать ситуацию в критический момент, и в то же время за каждую задачу отвечает конкретный программист.

Теперь о недостатках. Предполагаемая возможность смены оператора за терминалом означает тщательную проработку условий этой смены. Нет худшего варианта, чем сумбурное чередование лиц перед монитором. Если избрать такой вариант, на тренировках особое внимание следует уделить именно "смене караула". Приведем один из возможных вариантов организации этой смены. Программист доводит задачу до посылки жюри, после чего принимается за другую на бумаге, освобождая машину. При получении отрицательного ответа от жюри, программист, когда место за терминалом освободится (никого НЕ выгоняя!) сам (или советуясь с другими) решает, заняться ли ему отладкой не зачтенной задачи, или набирать новую. Если в результате принятого решения очередная посылка жюри происходит достаточно быстро (10-15 минут), то можно сделать оба дела, прежде, чем партнер-программист будет в состоянии приступить к своей работе за клавиатурой, а не на бумаге.

Подчеркнем, что каждому всегда есть чем заняться, т.е., простоя как такового не происходит. Если программист не занят на машине, он составляет тесты (в том числе, может помочь в составлении тестов партнеру, если необходимо "сменить рубашку").

В целом, работа происходит "попарно" - каждый программист с помощью математика сдает очередную задачу. Команда как бы состоит из четырех человек, в чем тоже некоторое преимущество.

7. Заключительные общие слова.

Повторим важную мысль, уже высказывавшуюся выше: для конкретной команды, скорее всего, ни одна из перечисленных стратегий не будет идеальной. Нужно отобрать то, что наиболее подходит, из каждой, и составлять собственную.

Однако, неверно поступать так: на нескольких тренировках пробовать то одно, то другое. Результаты реально зависят от большого числа факторов, а не только от выбранной стратегии, и после проведения этих тренировок команда окажется не в лучшем положении, чем до них.

Один из способов создать свою стратегию состоит в том, чтобы собравшись всей командой, обсудить известные каждому и сформулировать общие идеи, принципы. Далее, после каждой тренировки нужно разбирать проявившиеся недостатки, придумывать методы их устранения. Такой процесс способствует прояснению стратегии, уточнению деталей. Каждый участник привыкает делать свою работу, учится эффективному взаимодействию с другими членами команды.

Некоторые технические детали.

Помимо выработки стратегии общего поведения, есть много относительно "небольших" вопросов, возникающих при любой технике. Далее изложены некоторые мысли по этим моментам.

1. Использование программы "монитор".

"Монитор" - это программа, предназначенная для просмотра текущих результатов других команд во время тура. Монитор - это спасение и головная боль тренера и руководителя команды, с трепетом наблюдающих за его показаниями в продолжение соревнований. Однако, использование монитора участниками имеет и положительные, и отрицательные аспекты. К положительным моментам относится помощь в распознавании простых и сложных задач. Анализ того, какие задачи пытались решать лидеры, сколько попыток сдач зафиксировано среди первой десятки по той или иной задаче, может оказать немалую помощь в выборе очередной задачи для решения. Показания монитора очень часто - верный критерий сложности задачи. В результате просмотра команда сэкономит ресурсы, которые могли быть брошены на решение или исследование сложности "безнадежных" задач, правильнее определит очередность решения других задач.

Отрицательной стороной является сильный психологический шок, который испытывает человек, видя текущее положение своей команды в турнирной таблице. У команды возникает стремление сделать что-то быстрее, догнать и перегнать, часто приводящее, наоборот, к снижению производительности. Угнетающее воздействие, которое производит монитор на команду тем опаснее, что его крайне трудно, если вообще возможно смоделировать на тренировках.

Один из вариантов состоит в том, чтобы просматривать монитор после получения сообщения от жюри о засчитывании очередной задачи, начиная со второй. В этом случае помощь монитора при выборе очередной задачи может оказаться вполне реальной, а результаты работы команды будут, по крайней мере, не хуже при очередном просмотре по сравнению с предыдущим.

Если при просмотре команда увидит себя в верхней части таблицы, то эффект также может быть двояким. С одной стороны, прилив сил; с другой, - невольное желание расслабиться, посчитать, что победа уже близка, что главное сделано. Итак, программа монитора стоит того, чтобы о ней задуматься до того, как команда отправится на соревнования.

2. Отсечение созданных тестов.

Часто в процессе тестирования принимают участие два и более члена команды, а за клавиатурой находится только один из них. При этом часть тестов может создавать ему трудности при наборе (большой объем, сложные расчеты, необычный формат ввода), и в то же время, проверять лишь один из параметров программы (скажем, запас длины массива), в котором он уверен и без того. В этом случае оператор может отсечь такие "неудобные", "невыгодные" тесты, чтобы сберечь время. Он делает это на свою ответственность. Никто не пожалеет его, когда команда в результате

проиграет из-за подобной ошибки, но такой подход экономит время, тоже необходимое как воздух.

3. Замена оператора за терминалом.

Разные люди имеют обыкновение по-разному создавать программы. Они используют различную структуру отступов, стили создания классов и генерации имен переменных. У каждого есть свой любимый метод реализации быстрой сортировки или создания двусвязного списка. Вследствие описанного эффекта, смена оператора в команде влечет наличие расходов на подстройку среды программирования и исходного текста под его вкус. Обычно не приносит хороших результатов попытка написать программу одним человеком, а отладить другим. Однако, не каждый способен интенсивно и эффективно работать в продолжение пяти-шести часов. Резонным компромиссом кажется ведение каждой задачи одним оператором, но для разных задач операторы могут различаться.

4. Отсечение простых и сложных задач.

Разделение задач на простые и сложные необходимо, об этом говорит простая арифметика. Пусть время на решение простой задачи составляет t минут, а на решение сложной T минут. Если обе задачи будут решены, то на этой уйдет (t+T) минут. Однако, если команда начнет с простой задачи, то ее результат составит (2*t+T) минут, а если со сложной, то (t+2*T) минут. Кроме того, сложные задачи как правило, приносят команде больше штрафного времени, и если пытаться решать их на ранней стадии, это время войдет во все последующие результаты. Едва ли можно указать исчерпывающую методику разделения задач на простые и сложные. Попытаемся лишь выделить моменты, работающие более или менее часто.

Для решения одних задач алгоритм решения очевиден. Для решения других требуется предварительная работа математического аппарата - вывод рекуррентных формул, дифференцирование, численные формулы, комбинаторный подсчет, дополнительные построения, и прочее. Будем называть такие задачи "программистскими" и "математическими". Как правило, олимпиадные программистские задачи проще математических - в силу того, что многие участники "руками" знакомы с используемыми структурами данных, приемами использования этих структур; наконец, в команде вполне может быть выделен специальный мозг для решения математических задач. В то же время, относительно небольшое число людей в стране ежедневно имеют дело с математикой, из-за чего математические знания постепенно ими утрачиваются, переходят в пассивное состояние. Однако, несомненно, можно придумать программистскую задачу, которая требует для своего решения больше времени, чем математическая, так что это правило - не панацея.

Далее отдельно рассмотрим решение математических и программистских задач. В решении математических задач многое зависит от правильного определения темы, на которую составлена задача. По разным темам разработаны многочисленные советы, методы, и приемы, собранные в обширных руководствах по математике. Отошлем по этой части читателя к изданным сборникам задач по алгебре и геометрии, математическому анализу, и просто олимпиадных задач на логику мышления.

Что касается программистских задач, то ориентиром часто может служить порядок перебора. Порядки подразделяются таким образом: факториал - экспонента - степень

- куб - квадрат - n*log(n) - линейный - логарифм. Обычно более сложна задача, имеющая более высокий порядок перебора. Это правило также имеет исключения. Одно из наиболее очевидных - переборная задача, для решения которой годится алгоритм Форда-Беллмана может быть реализована быстрее, чем задача, где "побеждает" поиск в ширину или в глубину, но нужен ответ за n*log(n) в силу выданных жюри временных ограничений.

5. Сортировка задач одного класса.

Речь идет о задачах, которые "понятно как решать" после того, как условия прочитаны. То есть, видны алгоритмы решения "общего случая". Такие задачи могут, тем не менее, весьма различаться по количеству случаев особых, когда общая формула не годится. В этой ситуации команде желательно заняться задачей, минимальной по числу особенностей. При этом важно убедиться, что условие задачи понято правильно. Небрежность, допущенная при понимании условия задачи сейчас, обойдется втридорога. Во-первых, команда потратит ресурсы на разработку алгоритма, решающего не ту задачу; во-вторых, потраченное время войдет в результат пропорционально общему числу решенных задач; в-третьих, переделка задачи иногда может обойтись дороже, чем если все бросить и написать заново.

6. Способы пересдачи.

Всегда неприятно, если посланная жюри задача не была зачтена. Возникает момент выбора: или "добивать" эту задачу, или не отрываться от рабочего процесса решения. Точного ответа на этот вопрос неизвестно. Оценки времени для окончания работы и над той, и над другой задачей могут оказаться ошибочными.

Одним из принципов выработки своей позиции по малоисследованным вопросам является моделирование такой ситуации на тренировках. Постепенно накапливающийся опыт поможет принимать правильное решение, учитывая особенности обеих задач, состояние команды, остаток времени, и другие (даже неосознанные) факторы.

Некоторую помощь может оказать так называемая "математическая культура", если команда ей обладает. В данном случае важно такое проявление этой культуры, как осознание, из чего состоит разработанный алгоритм не в плане операторов, а в плане идей, как обрабатываются данные, к какому результату это приводит, и каким на самом деле должен быть ответ. Схожая ситуация возникает при попытке студента вычислить интеграл. Он приходит к преподавателю, слышит в ответ "неправильно", и в качестве объяснения получает производную - обратную операцию. То есть бывает, что показать неверность ответа просто, даже не разбираясь в его тонкостях. Регулярно сталкиваясь с этой проблемой, студент постепенно учится проверять ответ известными ему "простыми" способами самостоятельно, а впоследствии он начинает автоматически, без принуждения понимать, почему тот или иной путь решения обречен. Это и есть проявление математической культуры.

7.Построение контрпримеров.

Одним из элементов тренировок может стать практика в построении контрпримеров к собственным решениям и к решениям собратьев по команде. За тех, кто посчитает такую процедуру ненужной, контрпримеры построит жюри.

Помимо "корыстной" пользы в виде предоставления на суд жюри более надежных решений, опыт построения контрпримеров важен и для общего развития, как элемент формирования математической и логической культуры человека. Разбирая партии, контрпримеры строят теннисисты и шахматисты. Из построенного извлекается две пользы.

Во-первых, когда партия приводится к разобранной ситуации, разбор далее обрывается фразой вроде "мат в три хода". Сия метода - анализ сложных ситуаций через ранее разобранные более простые - усиливает игрока и человека в умении анализировать ситуацию вообще, в умении отделять зерна от плевел и накапливать опыт, учиться на чужих ошибках.

Во-вторых, в схожей с разобранной ситуации либо схожим же образом строится контрпример, либо решающий будет точно знать, что таким путем контрпример не построить, и не потратит на исследование разобранного пути построения примера время и силы.

Вторая польза кажется даже более значительной, так как для каждой ситуации обычно существует несколько близких, а потому, разобрав одну задачу, получаем оружие на несколько случаев.

8. Использование эвристик.

Эвристика - алгоритм, решающий часть задач из заданного класса, но не решающий (или неизвестно, решающий ли) задачу в общем виде. В умелых руках эвристика на олимпиаде может стать грозным оружием.

С одной стороны, эвристика может быть засчитана как решение задачи, так как задачи проверяются на наборах тестов. Однако, следует помнить, что жюри, возможно, рассматривало некоторые эвристики, и потому приготовленный набор тестов уже приготовил западню вашей. Тем не менее, в минуту отчаяния даже такое применение эвристики может снять напряжение и помочь найти разгадку к задаче.

С другой стороны, сочетая написание эвристики с грамотным тестированием, можно выяснить, в чем сложность задачи. Каждая задача имеет много сторон (время, память, сложность обработки единицы данных, сложность описанных в условии структур данных и реализации операций над ними), и не всегда с первого взгляда очевидно, в чем заключается проблема. Тестирование эвристики в такой ситуации позволит программисту ясно выделить основную сложность задачи и далее бороться именно с ней.

Наконец, знания об эвристиках помогут экономить время на их изобретение. Если в процессе разбиения задачи на подзадачи выделилась такая, для которой (это можно выяснить заранее) неизвестно полиномиального (или достаточно хорошего алгоритма), а известны только некоторые эвристики, то игрок может как задуматься над необходимостью другого способа решения исходной задачи, так и упростить себе проверку жизнеспособности текущего рабочего способа решения, промоделировав решение выделенной подзадачи одной из известных ему эвристик.

Отметим, что неграмотное применение эвристик неизбежно погубит команду. Ярким примером такого применения является попытка во что бы то ни стало сдать

конкретную задачу с помощью эвристики. Если познания жюри достаточно обширны (а чаще всего, так и бывает), то раз не прошедшая тесты эвристика будет обречена и в дальнейшем.

В качестве рабочей поговорки по этой главе приведем фразу, высказанную преподавателем С.В.Сизым на одном из праздников:

9. Принцип "последней минуты".

Как в футболе каждая команда стремится исполнить "гол престижа", так и на олимпиадах каждая команда стремится решить хотя бы одну задачу. Кроме этого принципа, существуют побочные традиции, также не приносящие в большинстве случаев призовых очков, но удовлетворяющие чувство собственного достоинства участников.

Одной из таких традиций является сдача на последних минутах хоть какой-нибудь задачи, пусть и не проведенной через тестирование и построение контрпримеров - за вас это сделает жюри.

Следствием такой традиции является замедление работы жюри ближе к концу тура, и участники уходят, не зная результатов последней проверки. Можно даже послать несколько версий одной задачи - вреда это уже не принесет, так как если задача не будет засчитана, то и штрафных очков за нее не начислят.

10. Принцип "блицкрига".

Продолжительная оборона психологически труднее, нежели азартное нападение. Из этой завалящей мудрости можно сделать забавное развлечение. Если команда достаточно сильна, можно попробовать получить значительный отрыв уже в дебюте, и тогда команду будет сложно догнать, так как соперники могут торопиться со сдачей, ошибаться при наборе исходных текстов и написании формул. Разумеется, что сказанное относится лишь к соперникам, активно пользующимся программой "монитор", о которой шла речь выше. Более того, даже и эти соперники вовсе не обязаны делать какие-либо ошибки при работе. Но своим отрывом вы предоставляете им все возможности ошибиться, отработать менее тщательно, найти брешь в командной дисциплине.

Один из способов получить такой отрыв заключается в следующем. В момент старта двое берут себе по простой задаче и реализуют их, каждый самостоятельно, быстро и практически без ошибок, один - сразу на компьютере, второй - сначала на бумаге, а потом (возможно, с помощью первого) - за терминалом. Все это время третий участник занимается, например, математической задачей или нарабатывает подходы к остальным задачам, в общем, создает плацдарм для дальнейшей работы. При успешной реализации такой идеи получается, что через 50 минут после старта готовы две задачи и наработки для дальнейшего решения. Если и дальше все пойдет гладко, догнать такую команду будет весьма непросто.

Однако, описанный подход содержит и большие опасности. А именно, если что-то не получается у любого из троих участников, то никто не в состоянии прийти к нему на помощь, и в результате происходит задержка, а то и срыв работы всей команды. Простои возможны как при решении первых двух задач или сразу после него, так и в

дальнейшем. Чтобы применять подобную тактику, команда должна быть очень хорошо готова; и даже в этом случае, риск велик. Несколько снизить риск может блестящее взаимопонимание между партнерами. Умение понимать друг друга с полуслова может выручить в критическую минуту.

На практике такой подход почти никогда не будет окупаться. Накладки, ошибки, индивидуальные промахи, непонимание условий, неверный выбор первых задач, неучтенные граничные случаи - очень много всего в природе противостоит приведенной методике. Но задуматься о существовании принципа "блицкрига" стоит уже потому, что откорректировав систему тренировок с его учетом, можно добиться улучшения взаимопонимания в команде, что пригодиться уже всегда.

11. Метод нисходящего проектирования.

Одним из самых дорогостоящих этапов программирования является отладка. С одной стороны, потому что на ее освоение традиционно отводится мало времени (и как следствие, по технологии отладки написано не так много литературы, как по кодированию вообще), с другой - так как временные затраты на отладку часто недооценивают, и считают, что программа "почти готова", когда успешно происходит трансляция.

Следствием этой ситуации является слабость владения участниками отладочными инструментами, существенные потери времени на "почти готовую" программу и не работающая программа в момент окончания тура. Этот и несколько следующих пунктов содержат некоторые советы, помогающие сократить время на отладку. Секрет в сокращении этого времени часто кроется в самой технологии программирования. Выражаясь метафорически, строители экономят время, сочетая изъятие почвы для котлована с монтажом крана.

Метод нисходящего проектирования призван сократить временные затраты на написание алгоритма и последующую отладку программы. Суть его, вкратце, заключается в следующем. Изначально в задаче выделяются некоторые "главные" подзадачи, объясняющие алгоритм в целом, но не обязательно "опускающиеся" до деталей этого алгоритма. Рассмотрим в качестве примера поиск в ширину в графе. Его обычно разбивают на три подзадачи: инициализация очереди вершин, совершение очередного шага (с анализом, не пора ли закончить работу) и построение кратчайшего пути.

После этого каждая подзадача решается тем же методом. Инициализация разбивается на создание пустой очереди и помещение в эту очередь стартовой вершины (иногда номер стартовой вершины приходится дополнительно вычислять), совершение очередного шага - на извлечение очередной вершины, ее обработке и занесении в очередь (более или менее эффективным образом) соседей вершины, а также, проверке на достижение цели; наконец, построение кратчайшего пути состоит в "обратном" проходе по самому графу. При необходимости, эти шаги конкретизируются далее, и так до тех пор, пока сформулированная подзадача не окажется доступной для простой и ясной записи на языке программирования.

Основная идея метода - не пытаться программировать сразу. Пошаговая детализация (программирование "сверху вниз") автоматически заставляет человека формировать понятную ему же структуру программы. После завершение трансляции (также

автоматически) формируется первичный набор тестов: каждый тест отлаживает конкретную подзадачу. Аккуратное проектирование обычно приводит к тому, что программист уверенно представляет себе работу каждой конкретной подзадачи, ее водные и выходные данные, и потому в состоянии протестировать именно ее. По окончании тестирования конкретной подзадачи, можно тестировать другие подзадачи независимо (!); эта независимость дает также возможность тестировать подзадачи по ходу реализации программы, генерируя после трансляции уже первично отлаженный код.

Наконец, упрощается и последующая отладка: при получении неверного результата программа может быть протрассирована, и проверка результата на очередном шаге сведется к пониманию, верно или неверно отработала очередная подзадача. Возможность независимой отладки подзадач сказывается и здесь, сужая область внесения возможных изменений трассируемыми подзадачами.

Помимо перечисленного, метод уменьшает нагрузку на мозг, ибо человеку не приходится решать проблему "что делать". У программиста автоматически формируется задание в каждый момент времени. Метод хорош также тем, что допускает множество конкретных воплощений и сочетается с разнообразными усовершенствованиями. Некоторые усовершенствования перечислены ниже.

12. Блочная отладка.

В таких языках программирования, как Си и Паскаль, имеется понятие блока. Прием блочной отладки заключается в рассмотрении каждого блока как отдельной подзадачи в момент написания программы. Как правило, программисты любят создавать небольшие по объему блоки, что способствует упрощению их отладки.

Такой прием весьма дисциплинирует программиста. Чем дольше прием применяется, тем больше блоков программист способен отлаживать непосредственно при их создании. С блоков он переходит на логически связные куски программы, на подпрограммы, вспомогательные алгоритмы, и т.п. Будучи уверен в "кирпичиках", их которых состоит программа, человек в дальнейшем тестирует уже "здание" в целом, не отвлекаясь на мелочи. Прием особенно эффективен в сочетании с нисходящим проектированием.

13. Выделение объектов и их свойств.

Основной целью двух предшествующих приемов было упростить понимание человеком структуры, логики программы - своего детища. Упрощение понимания происходило за счет выделения процессов, протекающих в программе, высвечивания их взаимодействия.

Можно улучшить понимание, рассмотрев происходящее с другой стороны. Каждая программа оперирует некоторыми данными, структурами данных. Более того, в мире, управляемом программой, обычно "живут" и взаимодействую конкретные представители этих структур: настоящие стеки, очереди, связные списки, сбалансированные деревья, контрольные суммы и другие забавные "существа", или "объекты данных" (в речи часто сокращаемые до термина "объекты").

Для каждого объекта известен набор операций, которые допустимо проводить над данным объектом ("набор методов объекта"). Программа будет работать правильно, если для каждого объекта корректно выполняются допустимые над ним операции, и все объекты правильно взаимодействуют, выполняя операции друг над другом.

Может оказаться, что в программе (или в реализуемой на очередном этапе подзадаче) весьма ограниченное число объектов, тогда, обеспечив корректность действий над объектами, программист сводит отладку (да и само программирование) к организации "поведения" нескольких субъектов по известным из условия задачи (и разработанного алгоритма) правилам. Это позволяет отделить друг от друга, сделать независимыми два этапа программирования: реализация объектов, операций над ними, и взаимодействия между объектами. Метафора: сначала программист создает молоток, отвертку, гвозди; а потом, пользуясь уже созданными инструментами, организует сборку шкафа.

14. Событийно управляемые системы.

Многие ошибки происходят в результате некорректных операций над определенными объектами: чтение и запись за границей массива, вычисления вне допустимого диапазона значений переменной, обращение к неиспользуемому участку памяти. Одним из предохранителей для таких ошибок служит написание событийно управляемых систем.

Сразу отметим, что сказанное не означает, что везде и всюду следует писать событийно управляемые программы. Но сама тренировка в их написании научит программиста лучше понимать и организовывать взаимодействие между объектами, в результате чего он будет допускать меньше ошибок во всех, а не только в событийно управляемых программах.

Как и выше, рассмотрим программу как некий абстрактный (сказочный) мир, где живут свои герои, с характером и заданным набором свойств (методов). Только теперь общение между героями будет происходить "цивилизованно". Вместо того, чтобы ударить по гвоздю, молоток пошлет ему сообщение "по тебе пора ударить". (Это сообщение может прочитать и еще кто-нибудь, но, за отсутствием полезной для себя информации, проигнорирует). Гвоздь проверит, не погнут ли он, находится ли он в досягаемости молотка, и пошлет стене сообщение о последствиях удара. Например, это сообщение может звучать так: "В точке с координатами (x,y) будет произведен удар силой P кг/см2." Стена примет это сообщение и отреагирует по-своему.

Явное объявление происходящего через сообщения имеет два преимущества. Во-первых, проверка возможности совершения действия над объектом поручается самому объекту (а кто, кроме самой функции знает, можно ли ее дифференцировать). Тем самым, эта проверка производится там, где максимально доступны ресурсы объекта - то есть, необходимые для этой проверки данные. Упрощается обработка некорректных ситуаций - объект вправе либо не реагировать на них, либо посылать ответное сообщение с рассказом о происшедшем. При моделировании параллельных процессов упрощается синхронизация. При происхождении некоторого события программа получает возможность адекватно отреагировать, так как откликнутся только те объекты, которых сообщение заинтересует (нет опасности, что в ответ на крик лыжника о помощи сойдет лавина). Во-вторых, упрощается отслеживание происходящего в программе. Нет больше нужды копаться в конкретных операторах

работы с ячейками памяти, достаточно просто выводить куда-либо в отладочном режиме посылаемые сообщения. Как правило, этой деловой переписки достаточно, чтобы установить "куда идут деньги".

Но часто разрабатываемые программы достаточно просты, и нет смысла палить из пушки по воробьям. Тем не менее, идеи событийной управляемости часто оказывают помощь и при реализации простых задач. Само понимание возможности такой работы учит программиста грамотно организовывать поведение объектов в абстрактном мире, избегая ошибок.

15. Граничные тесты.

Многие программы отлично работают на "средних" примерах, но (увы) неверно отрабатывают, когда один из (или несколько) параметров задачи принимают крайние значения. К сожалению, часто не представляется возможным отследить все комбинации крайних значений всех переменных, это число слишком велико. Чтобы как-то скрасить картину, используют следующие два приема.

Во-первых, на стадии разработки программы (написания, проектирования алгоритма, а не кода) выписываются параметры задачи и их границы. Для многих задач окажется, что число этих параметров не столь велико (не превосходит 10). Тогда в процессе тестирования возможно запустить тесты, содержащие все граничные значения всех параметров. Отметим, что речь здесь не идет о параметрах, существенно влияющих на время выполнения программы (как правило, олимпиадные программы должны работать достаточно быстро), так как иначе выполнение тестов займет слишком много времени. Кроме того, по смыслу задачи часто ясно, какой набор из 10-15 тестов достаточно прогнать, чтобы убедиться в правильности обработки граничных значений.

Во-вторых, можно использовать блочную отладку. Внутри одного логического блока программы редко встречается сразу большое количество объектов, и обработку граничных значений проверить несложно. Отметим, однако, что сама по себе правильная обработка граничных значений в каждом блоке еще не означает правильную их обработку программой в целом, так как ошибки в обработке могут быть и в логике программы.

16. Еще раз об integer, word и long.

Чтобы не допускать ошибок, связанных с преобразованием целых типов, можно воспользоваться следующим правилом:

Вся процедура проверки корректности выражения состоит из двух этапов: проверки того, что вычисляемое значение находится в диапазоне результата (int*int=long, abs(int)+abs(int)=word, но int+int=long (!!!)) и того, что автоматическое приведение типов в выражении не приводит к искажению значения (int*int+int - м.б., неверно, если первое слагаемое будет приведено к int; следует писать int*long+int. Обратите внимание, что int+int также может быть неверно, если транслятор сначала приведет результат к int, а потом - присвоит его.)

Используя приведенный принцип, можно застраховать себя от выполнения второго шага (возможно, использовать принцип следует несколько раз в одном выражении,

например: (int+int)+long может не отработать, если (int+int) будет автоматически приведено к int).

17. Оператор комментария.

Часто программисты недооценивают оператор комментария, что приводит к неоправданным затратам времени на отладку программы, а также к многократным пересдачам ее жюри. Происходит это потому, что человек забывает логику своей программы, и в дальнейшем тратит время на то, чтобы понять, где именно происходит требуемая обработка, или как именно работает тот или иной фрагмент.

Не случайно руководства по программированию выделяют как особую стадию производства программы этап проектирования, метод проектирования сверху вниз, структурное и объектно-ориентированное программирование. Все эти средства помогают программисту создать продукт, в большой степени рассказывающий о том, как он устроен и что делает.

Однако, еще со времен, когда не было никаких языков программирования, кроме машинных кодов, существует метод описывать, что происходит в программе, и это - комментарии. Понятная программа во многом похожа на книгу: она содержит разделы, главы, параграфы. В какой-то мере пропуск строки напоминает про абзац. Логично, чтобы главы и части имели названия, а логически самостоятельные части (напоминающие, к примеру, статьи в сборнике) - предисловие.

Сказанное в несколько меньшей степени относится к олимпиадным программам, так как они часто не столь велики и использование описанных выше приемов делает их текст достаточно прозрачным для опытного глаза. Но не следует забывать, что олимпиада - это всего лишь часть жизниИХТЂиХТЂи пр ,щсионалы пишут программы, размер которых измеряется в тысячах строк, и в результатах их работы (исходных текстах!!!) приходится разбираться другим профессионалам.

Каждый, кто выполнял какой-либо совместный проект, быстро уясняет себе, что написать комментарий быстрее, дешевле и проще, чем отвечать на вопросы партнера о том, что происходит там-то и там-то. А если человек занимается программированием постоянно (другие же, по мнению авторов, едва ли способны и на олимпиадах показать высокие результаты), то комментарий неизбежно всасывается к нему в кровь, впитывается в руки. Написать грамотный комментарий, помогающий, а не затуманивающий от читателя текста программы ее суть, несложно, если отнестись к этому ответственно.

Кроме того, комментарии служат не только для выявления логики программы, но это и удобное место для выделения ее необычных особенностей. Использование только четных индексов массива, описание особых ситуаций или действий в вырожденных случаях и огромное разнообразие других важных особенностей можно разместить в комментариях.

18. Построение формальной модели.

Разделение ролей в команде проповедуется не случайно. Его необходимость вытекает из следующего наблюдения. Если, прочитав условие, сесть и начать набирать исходный текст первого пришедшего в голову алгоритма, то часто в

результате получается длинная, неудобная для отладки, внесения изменений и использования в качестве вспомогательного алгоритма программа, в то время как после построения математической модели и записи алгоритма решения на ее основе все программирование сводится к набору нескольких строк кода.

Одним из примеров такого рода служит определение простоты числа. Можно написать программу, делящую заданное натуральное число на все, меньшие его натуральные числа. Несложное размышление подскажет, что достаточно делить на 2 и все нечетные числа для проверки простоты. Однако, подумав еще немного, можно сообразить, что достаточно делить на 2 и все нечетные числа, не превосходящие арифметического квадратного корня из заданного числа, ибо если есть множитель, больший корня, то должен быть и множитель, меньший корня. Уже для такого небольшого числа как миллион две последние версии программы проделают 500 тысяч сравнений и 500 сравнений соответственно.

19. Копирование блоков.

Одним из приемов ускорения набора текста является копирование уже набранного блока кода и небольшая его модификация в дальнейшем. Но в реальной жизни порой приходится корректировать один из этих блоков в дальнейшем и программист в этот момент обязан задать себе вопрос: как внесенное изменение отразится на всех других похожих блоках.

Такой вопрос предполагает, что программист в состоянии определить, где именно находятся похожие блоки, и чем именно эти блоки отличаются. Если применять копирование блоков бездумно, то ответы на эти вопросы могут занять больше времени, чем хотелось бы. Поэтому:

- Размещайте такие блоки невдалеке друг от друга.
- Комментируйте отличия каждого блока от других.

Может быть, если программист не вполне уверен в своем умении быстро ответить на приведенный вопрос, ему и вообще воздержаться от применения подобной практики, путем выделения общего во всех блоках в отдельную процедуру.

Смысл и частота тренировок.

1. Смысл тренировок.

Тренировки - это одно из немногих мест, где команда в состоянии почувствовать, чего же она стоит на самом деле, и где ее участники могут получить новые знания. Тренировки служат для передачи теории и опыта, для совместного обсуждения проблем, задач, решений, выработки умения работать в команде, для изучения новых аспектов и разделов дисциплины программирования.

Учитывая особенности соревнования, нужно подбирать и теоретический материал. Не претендуя на абсолютную истину все же заметим, что при подготовке к студенческой олимпиаде изучение программирования на машине Тьюринга или сетях Петри может принести больше пользы, чем асовое исполнение текстов на ассемблере.

Это не означает впрочем, что ассемблер изучать не нужно. Исходя из здравого смысла, можно предположить, что тренировки будут тем полезнее, чем шире будет их кругозор. Может быть, человеку проще думать и действовать, он более уверен в себе, если то, что приходится изобретать на соревновании (и, кстати, на работе - тоже) где-то было слышано, опробовано, имеется некоторый опыт.

Усвоенные на тренировках разнообразные идеи составляют арсенал участника для борьбы с задачами, его походный набор для изготовления программ. И чем больше этот набор, чем более совершенны его инструменты, тем качественнее будет и продукт.

2. Частота тренировок.

Личный опыт показывает, что максимальным сроком, когда человек еще способен помнить, о чем шла речь, является неделя. Поэтому прежде всего следует рассмотреть возможности устраивать тренировки хотя бы раз в неделю.

Если это невозможно, то кажется разумным увеличить самостоятельную нагрузку, создать способы промежуточных встреч/общения между участниками, между участниками и тренером, тренерами, преподавателями, консультантами.

3. Элементы тренировок.
(в порядке убывания важности).

3.1. Широта кругозора.

Большое число разнообразных идей, впитанных мозгом, позволяет ему находить что-то знакомое в самых необычных ситуациях. Кроме того, за время впитывания этих идей мозг, как правило, учится принимать решения в ситуациях нестандартных.

Обыденностью для участника становится задача "что делать, когда точного решения наизусть неизвестно". Жизнь и оргкомитет предлагают нам задачи, часто в чем-то необычные, неосвоенные, содержащие разнообразные препятствия на пути решения "обычными" методами. Знание нестандартных подходов, а также методов преодоления трудностей, позволит участнику справится с этими затруднениями.

3.2. Математические задачи.

Для повышения квалификации программиста часто весьма полезно решать именно математические задачи, в том числе, олимпиадные. Во-первых, потому что во многих стратегиях отчетливо выделена роль математика, во-вторых, потому что применение математических методов решения может облегчить программирование (см. тж. выше пункт "Построение формальной модели").

3.3. Отладка.

Вынесена на третье место потому, что весьма часто недооценивают именно время, которое тратится на отладку. В результате команда может оказаться у финиша с прекрасной, но не работающей программой. Никто этого не поймет и не простит.

3.4. Режим реального контроля.

Имеется ввиду отработка навыков командного взаимодействия, выработка дисциплины. Участники должны научится работать так, чтобы не мешать друг другу. В какой-то степени, этому помогает стратегия; но стратегия - это всего лишь инструмент; люди должны уметь ее использовать. Учатся они этому именно на тренировках.

Название абзаца выбрано таким потому, что важно не только, чтобы никто не простаивал, но и чтобы задачи решались компактно по времени. То есть, все могут быть заняты полезным трудом, но если этот труд сводится к тому, что каждый решает свою задачу и очень долго, то надо обдумать вопрос о реорганизации.

3.5. Реализация ссылочных структур.

Другая крайность - прекрасное знание теории решения тех или иных задач, но большое число ошибок в реализации. Яркий пример из этой области - реализация ссылочных структур данных (например, это могут быть стеки, очереди, деки, и т.п.). Многие умеют использовать эти средства (в том числе, весьма нетривиально), но при торопливой реализации легко ошибиться. Косвенная адресация ничего не простит, а транслятор не поможет - используя указатели вместо объектов, человек берет на себя смелость управлять памятью.

3.6. Взаимопонимание.

Речь идет об отработке ситуаций, когда по тем или иным причинам нормальная работа команды невозможна. Пример: задача не сдана с десятой попытки. Возможны и другие ситуации, когда нужно объяснить партнеру, что происходит, или, наоборот, помочь; когда надо принимать решение о совместной работе над одной задачей, и т.п. Короче, если отработать методы эвакуации заранее, меньше будет ущерб от пожара.

3.7. Общее число задач.

Если уровень команды достаточно высок, то сразу несколько предложенных задач могут показаться ей простыми. Если есть необходимость предусмотреть такую ситуацию, нелишне на тренировках отработать взаимодействие участников в таком

"ускоренном" режиме, так как схема работы может отличаться от "нормального" режима, когда на решение задач на бумаге уходит большее время.

3.8. Тайм-брейк.

Один из элементов работы - решение "последней" задачи. Эту задачу приходится решать в ускоренном темпе, а простой она, как правило, не является. Если решать последнюю задачу как все остальные задачи, то времени не хватит. Речь о том и идет, чтобы попытаться выгадать, за счет отбрасывания других задач, те "золотые" 30-40 минут, которых не хватает, чтобы закончить работу.

3.9. Замеры времени.

Наконец, когда освоена приведенная выше техника, можно приступать к отработке быстрого командного решения задач. Но помните: если Вы с этим поторопитесь, если указанные выше элементы (а возможно, и другие, которых мы не учли) не будут учитываться сознанием участников, то придется отрабатывать их на ходу, а это - нервы, споры и проволочки (ударение на последнем слоге).

4. Роль запасного игрока.

Запасной - это особая роль. С одной стороны, он позволяет не сорвать выступление, когда кто-то заболел; с другой - его имя остается в тени после соревнований.

Следует особое внимание уделять работе с запасным участником (участниками), в том числе, учитывая психологические аспекты их невыступления.

5. Спарринг-партнеры.

Чтобы оценивать свои успехи более реально, интересно на последнем этапе (когда перешли к замерам времени) соревноваться с другой командой. Кроме того, все другие элементы можно обсуждать, а идеями обмениваться. Чем больше команд тренируются совместно, тем больше идей они смогут рассказать друг другу.

"Совместно" - не означает в одной комнате. Можно просто вместе ходить на лекции, общаться в коридорах, а у терминала собираться в одиночку. Цель абзаца - просто подчеркнуть общую идею совместной работы и ее перспективы.

Советы руководителю команды.

1. "А счастье было так возможно..."

Нет никакого смысла горевать по поводу того, что "чуть-чуть" не успели. Так будет всегда, и независимо от результата. Вообще, не очень понятно, что можно обсуждать после соревнования. Во всяком случае, если задачи интересные, то тема обеспечена. Когда же она надоест, можно обсудить допущенные технические ошибки и обратить внимание на пользу, которую все проделанное принесет в жизни.

2. Соотношение приза и груза.

Особая в нашей стране проблема казенных денег заставляет написать о том, что руководитель команды (тренер, наставник, как угодно) обязан проявить администраторскую смекалку и сделать так, чтобы у участников голова ни о чем, кроме программирования не болела.

Техника, расписание тренировок, билеты, деньги - все это на совести руководителя; на вопросы, не связанные с программированием должно уходить минимальное время.

Полезно помнить, что у участников есть родители. Своевременное и полное их информирование о настоящем и будущем сэкономит и руководителю, и участникам много нервов.

Наконец, очень хорошо, если руководитель никогда не повышает голос, всегда выслушивает любого собеседника (и если прерывает, то очень вежливо и только из-за нехватки времени), и уж совсем никогда не впадает в панику. Он, и прежде всего он должен заразить всех уверенностью, что всегда что-то можно придумать, и демонстрировать это на деле, потому что неожиданных ситуаций в работе тренера возникает уйма.